AI for Business Research

Course Notes[*]

Instructor: Renyu (Philip) Zhang

Scribed by the students who took this course in Spring 2025.

August 28, 2025

Contents

[*] See `https://github.com/rphilipzhang/AI-PhD-S25` for more details about the 2025 offering of this course. The course notes are scribed by the following students: Yiquan Chao, Yuxin Chen, Qilin Huang, Yu Jiang, Ningfan Lai, Guohao Li, Keming Li, Mengyang Lin, Zhe Liu, Quan Long, Lin Ma, Xiqing Qin, An Sheng, Chuchu Sun, Jin Wang, Tao Wang, Yachong Wang, Di Wu, Xinyu Xu, Jiaci Yi, Zhengyun Yu, Bingqi Zhang, Shu Zhang, and Zizhou Zhang.

Chapter 1: Overview

1 Artificial Intelligence and Business Research

1.1 AI and the Evolution of Human Civilization

Artificial Intelligence (AI) is increasingly recognized as a defining force in the evolution of human civilization. Beyond its immediate technological applications, AI embodies a deeper shift in the way humans interact with information, make decisions, and shape their environments. As computational capabilities expand, the systems that harness these capabilities transform from specialized tools into general-purpose agents of innovation and discovery.

A pivotal insight into the development of AI comes from Richard Sutton's essay (Sutton, 2019). Sutton argues that the most profound advances in AI over the past seventy years have not stemmed from intricate, domain-specific models, but rather from general methods that scale with computation. Specifically, he contrasts two paradigms:

- Domain-specific approaches, which encode expert knowledge and often achieve early successes but are prone to plateauing.
- General, computation-intensive approaches, which initially may seem inefficient or simplistic, but ultimately surpass their specialized counterparts by leveraging large-scale computation and vast datasets.

This "bitter" realization challenges the traditional valorization of human-crafted expertise in modeling. It suggests that *scalability*, *adaptability*, and computational leverage are not merely engineering conveniences, but *fundamental scientific virtues* in the design of intelligent systems.

Implication for researchers. In the long run, methods that harness *generic computational principles* tend to dominate. Crafting elegant domain-specific models may be intellectually gratifying, but it is unlikely to match the transformative power of scalable, data-driven techniques.

1.2 AI for Business Research: Conceptual Foundations

Within the domain of business research, AI serves a dual role: it is both an object of study and a transformative methodological resource. Broadly speaking, AI contributes to business research in two principal ways:

- **Direct Application.** AI technologies can directly address substantive business research questions. For instance, predictive modeling can inform *operational decisions*, and *causal inference* techniques powered by machine learning can test hypotheses about managerial interventions.
- **Facilitative Enhancement.** AI can facilitate traditional business research by improving data acquisition, feature extraction, and analytical scalability. For example, natural language processing (NLP) can transform unstructured customer feedback into structured data suitable for econometric analysis.

Thus, AI is not merely a technical tool; it redefines the epistemic infrastructure of business research. It enables scholars to ask new types of questions, utilize previously inaccessible data modalities, and model systems of unprecedented complexity.

1.3 The AI Flywheel: Scaling Effects in Research and Practice

A crucial conceptual model in understanding AI's power is the AI Flywheel, a reinforcing cycle comprising three elements:

$$\text{Compute} \rightarrow \text{Data} \rightarrow \text{AI Models} \rightarrow \text{Business Applications} \rightarrow \text{More Data}$$

Each component reinforces the others:

- Increased computation enables more sophisticated model architectures.
- Enhanced models generate more valuable business applications.
- Wider adoption of AI applications generates larger and richer datasets.
- New datasets fuel further computational training.

This virtuous cycle suggests that impactful research strategies should align with environments where outcomes improve naturally with scale—both in terms of data volume and computational resources.

1.4 Key Distinctions: AI versus Traditional Statistical Methods

While traditional statistical methods often emphasize fixed sample inference and parametric rigor, AI methods prioritize scalability, adaptivity, and approximation under computational constraints. Several conceptual differences are noteworthy:

This distinction does not imply a dichotomy. Rather, it highlights a continuum: rigorous empirical business research increasingly draws on techniques from both traditions, depending on the nature of the question and the data environment.

Aspect	Traditional Statistics	Modern AI/ML
Objective	Estimation and inference under fixed designs	Generalization under scalable learning
Model Complexity	Prefer parsimony (Occam's Razor)	Embrace complexity if warranted by data and compute
Data Regime	Small to moderate datasets	Massive, high-dimensional datasets
Emphasis	Confidence intervals, hypothesis testing	Prediction accuracy, robustness, transferability

Table 1. Key Distinctions: AI versus Traditional Statistical Methods

1.5 Toward a Scalable Research Mindset

The lessons from AI development call for a recalibration of research strategies in business academia:

- **Favor generalizable over finely tuned models.** Methods that require delicate adjustment to specific datasets are less likely to survive in dynamic environments.
- **Anticipate scaling effects.** Choose research problems and methods that become easier or more powerful as data and computational resources grow.
- **Leverage computation as a scientific variable.** Treat computational capacity not as a constraint, but as an experimental dimension, akin to sample size in traditional statistics.

In this context, business researchers can align themselves with the fundamental drivers of AI success: generality, adaptability, and computational scalability.

2 Prediction Problems in Business Research

2.1 Why Do We Care About Prediction?

Prediction tasks are central to both academic inquiry and practical decision-making across a wide spectrum of fields, including *economics*, *political science*, *finance*, *healthcare*, and *operations management.*

Accurate predictions can directly inform and improve decision-making. In domains ranging from weather forecasting to cancer screening, predictive models act as the foundation for actionable strategies that affect individual and societal welfare.

Moreover, in modern causal inference, prediction plays an indispensable role. Methods such as causal machine learning(in Lecture 10), double machine learning (DML, in Lecture 11), honest trees, and matrix completion(A topic to be taught next year) fundamentally rely on accurately predicting counterfactual outcomes—that is, estimating what would have happened under alternative scenarios. We now outline a handful of illustrating examples:

Domain	**Specific Predictive Tasks**
Macroeconomic forecasts	GDP growth, unemployment rates, inflation, election outcomes
Policy evaluation	Impacts of tax changes, welfare programs, environmental regulations
Market behavior	Demand forecasts, asset price movements, consumer preference trends
Operational decisions	Inventory management, resource allocation, insurance underwriting, and a very long description of healthcare diagnosis that needs to wrap across multiple lines because it contains a lot of detail and specific conditions that are important to enumerate.

Table 2. Examples of Critical Predictive Tasks

2.2 The Logic of “Predict Then Decide”

In many decision-making scenarios, prediction precedes and guides the choice of action. This paradigm is particularly powerful when direct causal identification is difficult or costly.

The typical workflow involves:

- Predicting future states (e.g., demand levels, market responses).
- Optimizing actions based on predicted outcomes (e.g., pricing, inventory replenishment).

However, this approach hinges critically on the accuracy and stability of the predictive models.

2.3 When Do Predictions Fail to Matter?

Prediction efforts can become irrelevant or even harmful under the following conditions:

- **Non-substantive outcomes:** Predicting variables that have little strategic importance offers negligible value.
- **Low predictive accuracy:** Inaccurate predictions undermine decision quality and may introduce systematic biases.
- **Failure to predict counterfactuals correctly:** Especially in causal settings, predictions based on flawed assumptions—such as the violation of unconfoundedness (Conditional Independence Assumption, *CIA*) or lack of overlap—lead to misleading inferences.

Thus, effective predictive research must balance accuracy, causal interpretability, and strategic relevance.

2.4 Prediction Versus Estimation: A Broader Perspective

The distinction between prediction and estimation reflects fundamental epistemological differences:

- *Prediction* focuses on minimizing error in unseen instances, often tolerating model misspecification if generalization improves.
- *Estimation* emphasizes recovering true underlying parameters or causal relationships, often prioritizing identification rigor over predictive performance.

The two dimensions can be conceptually cross-classified as follows:

	Without Intervention	**With Intervention**
Focus on Specific Features/Effects	Descriptive analysis, measurement construction	Causal inference, applied microeconometrics
Focus on Outcomes	Predictive modeling, forecasting	Structural estimation, counterfactual simulation

Table 3. Prediction Versus Estimation

Chapter 2: Deep Learning Basics

3 ML Techniques and Gradient Descent

3.1 Machine Learning in the Modern Scientific Landscape

Machine Learning (ML) is increasingly regarded not merely as a subfield of computer science, but as a new scientific methodology for discovery, prediction, and decision-making across diverse domains. It embodies a fundamental epistemological shift: from theory-driven, deductive modeling to data-driven, inductive inference.

The rise of ML is catalyzed by three reinforcing forces:

- *Data revolution.* The explosion of digital data in scale and scope.
- *Computational advancements.* Dramatic increases in processing power and algorithmic sophistication.
- *Methodological innovation.* Development of scalable learning algorithms capable of handling high-dimensional, non-linear, and noisy environments.

Period	**Key Developments**	**Description**
1940s–1950s	Cybernetics and early AI	Early cybernetics(Wiener, 1948) emphasized feedback loops and adaptive behavior, laying the groundwork for machine adaptation.
1950s–1960s	Statistical Learning Theory	Kolgomorov and Vapnik pioneered frameworks for learning from data under uncertainty. Vapnik (1995) formalized ML problems as mathematical optimization.
1950s–1980s	Connectionism and neural networks	Early models like neuron(McCulloch and Pitts, 1943) and perceptron(Rosenblatt, 1958) introduced biologically inspired learning systems. Progress stalled due to computational limitations ("AI winters").
1970s–1990s	Decision Trees and Ensemble Methods	Quinlan's ID3 algorithm (Quinlan, 1986) and Random Forests(Breiman, 2001) introduced practical, interpretable models for structured data learning.
1990s–present	Modern Machine Learning Renaissance	Availability of large datasets and computational resources led to mainstream adoption of ML. Breakthroughs in deep learning, boosting (e.g., AdaBoost(Freund and Schapire, 1997)), and generative models.

Table 4. A Historical Perspective on Machine Learning

In the realm of business research, ML empowers scholars to address complex, dynamic, and *heterogeneous systems* that traditional econometric techniques may inadequately capture.

3.2 Workflow of Supervised Learning

Modern machine learning encompasses several major paradigms, each addressing different types of problems:

- **Supervised Learning.** Learning a mapping from inputs X to outputs Y based on labeled examples (e.g., image classification, demand forecasting);
- **Unsupervised Learning.** Discovering patterns or structure within unlabeled data (e.g., clustering, dimensionality reduction).
- **Reinforcement Learning.** Learning policies for sequential decision-making via interactions with an environment (e.g., game-playing agents, adaptive pricing).
- **Generative Learning.** Modeling data distributions to generate new synthetic samples (e.g., text generation, image synthesis).

Among these, supervised learning forms the backbone of most predictive modeling tasks in business applications and will be the initial focus of our exploration. Typically, the data in a supervised learning context consists of input-output pairs:

$$(X_i, Y_i), \quad i = 1, 2, \ldots, n, \tag{1}$$

where $X_i \in \mathbb{R}^d$ represents the feature vector and Y_i represents the corresponding label or response.

We assume there exists an underlying true relationship between X and Y:

$$Y = f(X) + \epsilon, \tag{2}$$

where f is an unknown deterministic function, and ϵ is a random noise term with mean zero and finite variance.

General Objective. The goal of supervised learning is to construct an estimator $\hat{f}(X)$ that approximates the true function $f(X)$ as accurately as possible.

A common criterion for measuring predictive performance is the **mean squared error** (MSE):

$$\text{MSE} = \mathbb{E}\left[(Y - \hat{f}(X))^2\right], \tag{3}$$

where the expectation is taken over the joint distribution of (X, Y).

Classification *versus* Regression. Supervised learning tasks are broadly categorized into classification and regression, depending on the nature of the output variable Y:

- **Classification:** The output Y is categorical, typically taking values in a finite discrete set (e.g., $\{0, 1\}$ for binary classification, or more generally $\{1, 2, \ldots, K\}$ for K-class classification). The objective is to make a prediction $\hat{Y}$ of the correct class label Y for a given input X. The commonly used metrics include:
  - **Zero-One Loss**: $L(Y, \hat{Y}) = \mathbb{I}(Y \neq \hat{Y})$, where $\mathbb{I}(\cdot)$ is the indicator function;
  - **Cross-Entropy Loss**: $L(Y, \hat{p}) = -\sum_{k=1}^{K} \mathbb{I}(Y = k) \log(\hat{p}_k)$, where $\hat{p}_k$ is the predicted probability of class k.
- **Regression:** The output Y is continuous, typically a real-valued variable. The objective is to predict a real number $\hat{Y}$ as accurately as possible based on the input X. The commonly used metrics include:
  - **Squared Error Loss**: $L(Y, \hat{Y}) = (Y - \hat{Y})^2$;
  - **Absolute Error Loss**: $L(Y, \hat{Y}) = |Y - \hat{Y}|$.

The distinction between classification and regression influences the choice of models, loss functions, evaluation metrics, and theoretical analysis throughout the supervised learning pipeline.

Supervised Learning Pipeline. The typical supervised learning process consists of the following steps:

- **Model specification:** Choose a class of models to approximate $f(X)$, such as linear functions, decision trees, or neural networks.
- **Loss function selection:** Define a loss function $L(Y, \hat{f}(X))$ that quantifies the penalty for prediction errors. Common choices include:
  - Squared loss $L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2$ for regression tasks;
  - Cross-entropy loss $L(Y, \hat{f}(X)) = -[Y \log \hat{f}(X) + (1 - Y) \log(1 - \hat{f}(X))]$ for classification tasks.
- **Optimization:** Choose an optimization method to minimize the empirical risk, defined as the average loss over the training data:

$$\min_{\hat{f}} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, \hat{f}(X_i)). \tag{4}$$

  Examples include:
  - Ordinary Least Squares (OLS) for linear regression;
  - Gradient descent methods for more complex models.

- **Model selection and evaluation:**
  - Fit the model using the training dataset;
  - Evaluate its generalization performance using a separate testing dataset, typically by computing metrics such as the mean squared error (MSE), accuracy, precision, recall, or area under the ROC curve (AUC).

3.3 Model Selection and Evaluation

Model selection and evaluation are critical steps in the supervised learning pipeline. A good model should not only fit the observed training data well but also generalize effectively to unseen data. Typically, a model that performs exceptionally well on the training data may exhibit unstable performance on unseen data (overfitting), whereas a model that performs poorly on the training data will consistently underperform on unseen data (underfitting).

This trade-off between fitting the training data and generalizing to new data is known as the bias-variance trade-off, which is summarized in the following figure.

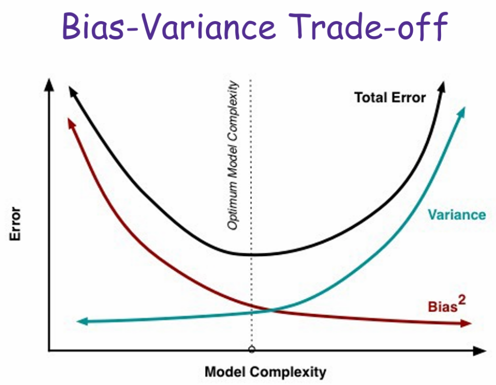


Figure 3.3.1. Bias-Variance Tradeoff

Formally, consider the Mean-Squared Error conditioned on X, the Bias-Variance trade-off is expressed as follows,

$$\underbrace{E_{\mathcal{D}}[((Y-\hat{f}(X,\mathcal{D}))^2]}_{\text{Error Conditioned on X}} = \underbrace{(E_{\mathcal{D}}[\hat{f}(X,\mathcal{D})]-f(X))^2}_{\text{Bias}} + \underbrace{E_{\mathcal{D}}[(E_{\mathcal{D}}[\hat{f}(X,\mathcal{D})]-\hat{f}(X,\mathcal{D}))^2]}_{\text{Variance}} + \underbrace{V(\epsilon)}_{\text{Noise}} \tag{5}$$

, and the total mean squared error will be

$$\underbrace{E_{X,\mathcal{D}}[(Y-\hat{f}(X,\mathcal{D}))^2]}_{\text{Mean Squared Error}} = E\{Bias_{\mathcal{D}}[\hat{f}(X,\mathcal{D}) + Variance_{\mathcal{D}}[\hat{f}(X,\mathcal{D})]\} + V(\epsilon) \tag{6}$$

, that is, the total mean squared error of a model is split into two parts: the bias and the variance.

In the face of the tradeoff, we are expected to select the model that optimally balances bias and variance to minimize the expected prediction error. To achieve this goal empirically, we need some standard methodologies for estimating a model's out-of-sample performance and selecting the best among competing models.

3.3.1 Training, Validation, and Testing Splits

A conventional approach is to partition the available dataset $\mathcal{D}$ into three disjoint subsets:

- **Training set** ($\mathcal{D}_{\text{train}}$): Used to fit candidate models.
- **Validation set** ($\mathcal{D}_{\text{val}}$): Used to tune hyperparameters and select among models.
- **Test set** ($\mathcal{D}_{\text{test}}$): Used only once, to report the final model's generalization performance.

The typical workflow is as follows:

1. Fit multiple candidate models $\{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_L\}$ using $\mathcal{D}_{\text{train}}$.
2. Evaluate each model on $\mathcal{D}_{\text{val}}$ by computing the average loss:
$$\hat{e}_l = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(X_i, Y_i) \in \mathcal{D}_{\text{val}}} L(Y_i, \hat{f}_l(X_i)). \tag{7}$$
3. Select the model with the smallest validation error:
$$\hat{f}^* = \arg\min_l \hat{e}_l. \tag{8}$$
4. Finally, assess the selected model's generalization error on the test set:
$$\hat{e}_{\text{test}} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(X_i, Y_i) \in \mathcal{D}_{\text{test}}} L(Y_i, \hat{f}^*(X_i)). \tag{9}$$

The test set must not be used during training or model selection. It should remain unseen until the final evaluation to avoid bias.

3.3.2 Cross-Validation

When the size of the dataset is small, reserving a separate validation set may not be feasible. In such cases, **cross-validation** provides an efficient way to estimate the model performance.

The most common variant is k**-fold cross-validation**. For each candidate model, we can evaluate its performance via the following procedure.

Algorithm 1 k-Fold Cross-Validation Procedure

```
Require: Dataset $\mathcal{D}$, number of folds $k$
Randomly partition $\mathcal{D}$ into $k$ approximately equal-sized subsets $\mathcal{D}_1, \dots, \mathcal{D}_k$.
for $j = 1$ to $k$ do
    Train the model on $\mathcal{D} \backslash \mathcal{D}_j$.
    Evaluate the model on the validation fold $\mathcal{D}_j$ to obtain validation error $\hat{e}_j$.
end for
Compute the cross-validation error:
    $$\hat{e}_{\text{CV}} = \frac{1}{k} \sum_{j=1}^{k} \hat{e}_j.$$
return Cross-validation error $\hat{e}_{\text{CV}}$
```

Cross-validation provides a more robust estimate of out-of-sample error, particularly when data is limited.

Time-series Cross Validation Time-series cross-validation is a model evaluation technique tailored for sequential data, where observations are ordered over time. Unlike standard *k-fold* cross-validation that randomly partitions data, time-series cross-validation preserves temporal order to prevent information leakage from future to past. The most common approach is the rolling or expanding window method, where the model is trained on an initial segment and evaluated on subsequent time steps, gradually incorporating more data. This technique ensures that predictions mimic real-world forecasting scenarios and provides a robust assessment of model performance over time.

3.3.3 Common Evaluation Metrics

Depending on the task, different metrics are used to evaluate models:

Table 5. Metrics and Their Calculation Formulas

Metric	**Calculation Formula**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall (Sensitivity)	$\frac{TP}{TP+FN}$
F1 Score	$2 \times \frac{Precision \times Recall}{Precision + Recall}$
Specificity	$\frac{TN}{TN+FP}$
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$
False Negative Rate (FNR)	$\frac{FN}{FN+TP}$
True Positive Rate (TPR)	$\frac{TP}{TP+FN}$ (Same as Recall)
True Negative Rate (TNR)	$\frac{TN}{TN+FP}$ (Same as Specificity)
Area Under Curve (AUC)	Integral of ROC curve (TPR vs FPR)
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{i=1}^{n} \lvert y_i - \hat{y}_i \rvert$
Mean Squared Error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
R-squared (R^2)	$1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$

, where the TP, TN, FP, FN represents true positive, true negative, false positive and the false negative respectively. The choice of metric should align with the practical objectives and costs associated with prediction errors.

3.4 Supervised Learning Techniques

We now introduce several fundamental supervised learning techniques. Each technique will be accompanied by the corresponding coding example for better understanding. The coding examples listed in the section can also be found in the jupyter notebooks that will be discussed in the section 3.6. A more detailed review of the core mathematical concepts upon which these techniques rely, including linear algebra, probability theory, and optimization, can be found in the appendix.

ML Technique	Core Mathematics Involved	Key Concepts
Logistic Regression	Convex optimization, probability	Logistic loss, likelihood maximization
Linear Regression	Linear algebra, optimization	Normal equations, least squares
K-Nearest Neighbors (KNN)	Euclidean geometry, distance metrics	Instance-based learning, majority voting
Decision Trees	Combinatorics, greedy algorithms	Impurity reduction, recursive partitioning
Random Forests	Probability (bootstrap theory), ensemble theory	Variance reduction, aggregation
Gradient Boosting	Functional approximation, optimization	Additive models, gradient descent on functions
Bagging (Bootstrap Aggregation)	Probability, resampling theory	Variance reduction, bootstrap sampling, averaging
Support Vector Machines (SVM)	Convex optimization, geometry	Max-margin hyperplane, duality
Deep Neural Networks(DNN) [1]	Multivariate calculus, optimization	Backpropagation, non-convex optimization

Table 6. Core mathematical concepts involved in several machine learning techniques.

3.4.1 *Linear Regression: Housing Prices*

Linear regression is the canonical model for supervised learning when the target variable Y is continuous. It assumes a linear relationship between the input features $X \in \mathbb{R}^d$ and the outcome $Y \in \mathbb{R}$:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_d X_d + \epsilon, \tag{10}$$

where ϵ is an independent, zero-mean random error term. In matrix notation, for a dataset of n observations, this can be written compactly as:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{11}$$

where $\mathbf{X} \in \mathbb{R}^{n\times(d+1)}$ includes a column of ones for the intercept. The objective is to estimate the parameter vector $\boldsymbol{\beta}$ by minimizing the sum of squared errors, that is, the ordinary least squares(OLS):

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(Y_i - \mathbf{X}_i^\top \boldsymbol{\beta} \right)^2 . \tag{12}$$

The closed-form solution is given by:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^\top \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{Y}, \tag{13}$$

assuming $\mathbf{X}^\top \mathbf{X}$ is invertible.

Linear regression models the relationship between a continuous dependent variable and one or more independent variables by fitting a linear equation to observed data.

Example. Suppose we want to predict the sale price of a house based on its living area (in square feet).

The linear model is:

$$\text{Price} = \beta_0 + \beta_1 \times \text{LivingArea} + \epsilon,$$

where ϵ captures random noise. We use a simple synthetic dataset, with:

- LivingArea (sqft): X
- Price (USD): Y

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(42)
X = 2.5 * np.random.randn(100, 1) + 1.5 # Living Area
y = 200000 + 50000 * X.flatten() + np.random.randn(100) * 20000 # Price

# Fit a linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict
X_test = np.linspace(min(X)[0], max(X)[0], 100).reshape(-1, 1)
y_pred = model.predict(X_test)

# Plot
plt.scatter(X, y, color=‘blue’,␣label=‘Data␣Points’)
plt.plot(X_test, y_pred, color=‘red’,␣linewidth=2,␣label=’‘itted Line’)
plt.xlabel(‘Living␣Area␣(1000␣sqft)’)
plt.ylabel(‘Price (USD)’)
plt.title(‘Housing␣Price␣Prediction␣via␣Linear␣Regression’)
plt.legend()
plt.show()
```

, where β_0 (intercept) estimates the baseline price when LivingArea = 0, and β_1 (slope) estimates the additional price per unit increase in living area.

3.4.2 *Logistic Regression: Predicting Customer Churn*

Logistic regression models the probability that a binary outcome $Y \in \{0, 1\}$ occurs, given input features X. Instead of modeling Y directly, it models the conditional probability using the logistic (sigmoid) function:

$$\mathbb{P}(Y = 1|X) = \frac{1}{1 + \exp\left(-(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d)\right)}. \tag{14}$$

The parameters $\boldsymbol{\beta}$ are estimated by *maximum likelihood estimation* (MLE), seeking to maximize the likelihood function:

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left(\mathbb{P}(Y_i = 1|X_i)\right)^{Y_i} \left(1 - \mathbb{P}(Y_i = 1|X_i)\right)^{1-Y_i}, \tag{15}$$

or equivalently, minimize the negative log-likelihood (logistic loss):

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} - \sum_{i=1}^{n} \left[Y_i \log \mathbb{P}(Y_i = 1|X_i) + (1 - Y_i) \log(1 - \mathbb{P}(Y_i = 1|X_i))\right]. \tag{16}$$

This optimization is convex and is typically solved via gradient descent or Newton-Raphson methods.

Example. Suppose we want to predict whether a customer will churn (i.e., cancel their service) based on their usage duration.

The features and label are:

- **X**: Average monthly usage (in hours)
- **Y**: 1 if the customer churned, 0 otherwise

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Generate synthetic data
np.random.seed(42)
X = 2.5 * np.random.randn(100, 1) + 5.0 # Average monthly usage
y = (X.flatten() < 5.0).astype(int) # Customers with low usage more likely to
    churn

# Fit a logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Predict probability
X_test = np.linspace(min(X)[0], max(X)[0], 100).reshape(-1, 1)
y_prob = model.predict_proba(X_test)[:, 1]
```

```

# Plot
plt.scatter(X, y, color=’blue’, label=’Data␣Points’)
plt.plot(X_test, y_prob, color=’red’, linewidth=2, label=’Predicted␣
    Probability’)
plt.xlabel(’Average␣Monthly␣Usage␣(Hours)’)
plt.ylabel(’Probability␣of␣Churn’)
plt.title(’Customer␣Churn␣Prediction␣via␣Logistic␣Regression’)
plt.legend()
plt.show()
```

, where the logistic function maps the linear score to a probability between 0 and 1. A negative β_1 implies that higher usage decreases the probability of churn.

3.4.3 K-Nearest Neighbors: Classifying Fruits by Weight and Color

K-Nearest Neighbors (k-NN) is a simple, non-parametric machine learning algorithm used for both classification and regression tasks. In classification, the method assigns a new observation to the majority class among its k closest training samples in the feature space, where distance is typically measured using metrics like Euclidean or Manhattan distance.

Given a query point x, its predicted class $\hat{Y}(x)$ is:

$$\hat{Y}(x) = \text{majority\_vote}\left(Y_i : x_i \in \mathcal{N}_k(x)\right), \tag{17}$$

where $\mathcal{N}_k(x)$ denotes the set of k training points nearest to x under a distance metric, typically the Euclidean distance, that is, $d(x, x') = \sqrt{\sum_{j=1}^{d}(x_j - x'_j)^2}$.

Unlike parametric methods(e.g. LR and Logit. R.), k-NN does not explicitly estimate model parameters. It memorizes the training data and defers computation until prediction time.

Theory Insights.

- **Bias-Variance Tradeoff.** A small k leads to low bias but high variance; a large k smooths the decision boundary, reducing variance at the cost of increased bias.
- **Curse of Dimensionality.** As the number of dimensions d increases, the volume of the feature space grows exponentially. Consequently, the number of samples n required to maintain a fixed density grows exponentially with d, making nearest neighbor methods less effective in high dimensions.
- **Computational Complexity.**

  - Training complexity: $O(1)$ (lazy learning, just store data).
  - Prediction complexity: $O(n \times d \times k)$ (must compute distance to every training point).

- **Distance Metric Sensitivity.** Proper feature scaling (e.g., standardization) is crucial because distance-based methods are sensitive to feature magnitudes.

Example. Suppose we want to classify whether a fruit is an apple ($Y = 0$) or an orange ($Y = 1$) based on two features:

- $\mathbf{X}_1$: Weight (grams)
- $\mathbf{X}_2$: Color intensity (darker orange color has higher values)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# Generate synthetic data
np.random.seed(42)
weight = 100 + 30 * np.random.randn(50) # Apples around 100g
color = 50 + 10 * np.random.randn(50) # Apples lower color intensity
X_apples = np.column_stack((weight, color))

weight = 150 + 30 * np.random.randn(50) # Oranges around 150g
color = 80 + 10 * np.random.randn(50) # Oranges higher color intensity
X_oranges = np.column_stack((weight, color))

X = np.vstack((X_apples, X_oranges))
y = np.array([0]*50 + [1]*50) # 0=Apple, 1=Orange

# Fit a k-NN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)

# Predict on a grid for visualization
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-10, X[:,0].max()+10, 100),
                     np.linspace(X[:,1].min()-10, X[:,1].max()+10, 100))
grid = np.c_[xx.ravel(), yy.ravel()]
probs = knn.predict_proba(grid)[:, 1].reshape(xx.shape)

# Plot
```

```
plt.contourf(xx, yy, probs, levels=25, cmap="RdYlBu", alpha=0.6)
plt.scatter(X_apples[:,0], X_apples[:,1], color='red', label='Apples')
plt.scatter(X_oranges[:,0], X_oranges[:,1], color='orange', label='Oranges')
plt.xlabel('Weight␣(grams)')
plt.ylabel('Color␣Intensity')
plt.title('Fruit␣Classification␣via␣k-NN')
plt.legend()
plt.show()
```

, where the k-NN classifier predicts the label of a query fruit by majority voting among its k nearest neighbors. In this case, heavier fruits with higher color intensity are more likely to be classified as oranges.

3.4.4 Decision Trees: Predicting Loan Approval

Decision trees are non-parametric supervised learning models used for classification and regression tasks. The core idea is to recursively partition the feature space into disjoint regions by making a sequence of hierarchical decisions.

At each internal node, the data is split according to a feature X_j and a threshold s:

$$\text{If } X_j \leq s, \quad \text{go left;} \quad \text{else, go right.} \tag{18}$$

The goal is to find the feature and threshold that yield the largest reduction in ”impurity”.

Impurity Measures. The impurity of a node can be measured in various ways, depending on the task:

- **Classification:**
  - Gini index:
$$G(p) = \sum_{k=1}^{K} p_k(1 - p_k), \tag{19}$$
    where p_k is the proportion of class k samples in the internal node.
  - Entropy:
$$H(p) = -\sum_{k=1}^{K} p_k \log(p_k). \tag{20}$$
- **Regression:**
  - Mean Squared Error (MSE) within the internal node.

The best split is chosen to maximize the impurity reduction:

$$\begin{aligned}\Delta\text{Impurity} =&\text{Impurity(parent node)}\\ &-\left(\frac{n_{\text{left node}}}{n_{\text{parent node}}}\text{Impurity(left node)}+\frac{n_{\text{right node}}}{n_{\text{parent node}}}\text{Impurity(right node)}\right).\end{aligned} \tag{21}$$

Theory Insights.

- **Overfitting.** Decision trees tend to overfit if grown too deep. Pruning techniques (e.g., limiting max depth or minimum samples per leaf) are used to control complexity.
- **Bias-Variance Tradeoff.** Shallow trees have high bias and low variance; deep trees have low bias and high variance.
- **Greedy Construction.** Finding the globally optimal tree is NP-complete. Practical algorithms build trees greedily by locally optimizing impurity reduction at each step.
- **Interpretability.** Decision trees are highly interpretable and can naturally handle both numerical and categorical features.

Example. Suppose we want to predict whether a customer's loan application will be approved based on two features:

- **X**$_1$: Applicant's annual income (in $1000s)
- **X**$_2$: Applicant's credit score
- **Y**: 1 if loan approved, 0 otherwise

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# Generate synthetic data
np.random.seed(42)
income = 50 + 30 * np.random.randn(100) # Income in $1000s
credit_score = 600 + 100 * np.random.randn(100) # Credit score
X = np.column_stack((income, credit_score))
y = (income + credit_score/10 > 110).astype(int) # Approval if income + (
    credit_score/10) > threshold
```

```
# Fit a Decision Tree
clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X, y)

# Visualize Decision Boundary
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-10, X[:,0].max()+10, 200),
                     np.linspace(X[:,1].min()-50, X[:,1].max()+50, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)

plt.contourf(xx, yy, probs, levels=np.linspace(0,1,20), cmap="RdYlBu", alpha
    =0.6)
plt.scatter(X[:,0], X[:,1], c=y, cmap="bwr", edgecolor=’k’)
plt.xlabel(’Annual␣Income␣($1000s)’)
plt.ylabel(’Credit␣Score’)
plt.title(’Loan␣Approval␣Prediction␣via␣Decision␣Tree’)
plt.show()

# Optional: Visualize the Tree Structure
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=["Income", "CreditScore"])
plt.title(’Decision␣Tree␣Structure’)
plt.show()
```

The decision tree partitions the feature space into rectangular regions based on applicant’s income and credit score thresholds. A deeper tree could capture more nuances but risks overfitting the training data.

3.4.5 Random Forests: Predicting Credit Scores

Random Forests are ensemble learning methods that combine multiple decision trees to improve predictive performance and mitigate overfitting.

A random forest constructs B decision trees $\{T_b\}_{b=1}^{B}$ by:

- Drawing a bootstrap sample (with replacement) from the training data for each tree.
- At each split, selecting a random subset of m features and choosing the best split only among those features instead of all the features.

The final prediction aggregates the outputs of all trees:

$$\hat{Y}(x) = \begin{cases} \text{majority\_vote}\left(T_1(x), T_2(x), \dots, T_B(x)\right), & \text{for classification} \\ \frac{1}{B}\sum_{b=1}^{B} T_b(x), & \text{for regression} \end{cases} \tag{22}$$

Theory Insights.

- **Variance Reduction.** Bagging (bootstrap aggregation) reduces variance without increasing bias significantly.
- **De-correlation of Trees.** Random feature selection ensures that individual trees are less correlated, making the ensemble stronger.
- **Out-of-Bag (OOB) Estimation.** Samples not included in a bootstrap sample can be used as a validation set, providing an unbiased estimate of test error without needing a separate hold-out set.
- **Feature Importance.** Random forests can estimate the importance of each feature by measuring the decrease in impurity or prediction error when that feature is used for splitting.

Example. Suppose we want to predict whether a customer has a high or low credit score based on two features:

- $\mathbf{X}_1$: Customer's total outstanding debt (in \$1000s)
- $\mathbf{X}_2$: Customer's annual income (in \$1000s)
- **Y**: 1 if high credit score, 0 otherwise

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

# Generate synthetic data
np.random.seed(42)
debt = 30 + 20 * np.random.randn(200) # Outstanding debt
income = 60 + 25 * np.random.randn(200) # Annual income
X = np.column_stack((debt, income))
y = ((income - debt) > 40).astype(int) # Higher income vs debt -> better
    credit

# Fit a Random Forest
```

```
rf = RandomForestClassifier(n_estimators=100, max_depth=5, oob_score=True,
    random_state=42)
rf.fit(X, y)

# Predict on a grid
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-10, X[:,0].max()+10, 200),
                     np.linspace(X[:,1].min()-10, X[:,1].max()+10, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
probs = rf.predict_proba(grid)[:, 1].reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, probs, levels=np.linspace(0,1,20), cmap="RdYlBu", alpha
    =0.6)
plt.scatter(X[:,0], X[:,1], c=y, cmap="bwr", edgecolor='k')
plt.xlabel('Outstanding␣Debt␣($1000s)')
plt.ylabel('Annual␣Income␣($1000s)')
plt.title('Credit␣Score␣Prediction␣via␣Random␣Forest')
plt.show()

# Show Out-of-Bag score
print(f"Out-of-Bag␣(OOB)␣estimate␣of␣accuracy:␣{rf.oob_score_:.3f}")

# Feature Importance
importances = rf.feature_importances_
print(f"Feature␣Importances:␣Debt={importances[0]:.3f},␣Income={importances
    [1]:.3f}")
```

, where the random forest aggregates predictions across many decorrelated decision trees, providing a smoother decision boundary and reducing overfitting compared to a single tree. The Out-of-Bag (OOB) score gives a reliable estimate of the model's generalization error without needing an explicit validation set.

3.4.6 Gradient Boosting: Predicting Purchase Intent

Gradient Boosting is an ensemble method that builds a strong learner by sequentially adding weak learners, typically shallow decision trees, to correct the errors made by the ensemble thus far.

Starting with an initial constant prediction $F_0(x)$, Gradient Boosting updates the model iteratively:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x), \tag{23}$$

where $h_m(x)$ is the base learner (a small decision tree) fitted to the negative gradient of the loss function, and γ_m is the learning rate or step size.

Mathematical Procedure. At each stage m:

1. Compute the pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(Y_i, F(X_i))}{\partial F(X_i)}\right]_{F=F_{m-1}} \quad \text{for all } i = 1, \ldots, n, \tag{24}$$

   where $L(Y, F(X))$ is the specified loss function (e.g., squared error for regression, logistic loss for classification).

2. Fit a base learner $h_m(x)$ to predict the pseudo-residuals r_{im}.

3. Find the optimal step size γ_m:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(Y_i, F_{m-1}(X_i) + \gamma h_m(X_i)\right). \tag{25}$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x). \tag{26}$$

Theory Insights.

- **Bias Reduction.** Boosting focuses on reducing bias by sequentially improving the fit to the data.
- **Overfitting Control.** Shrinking the learning rate γ_m and limiting tree depth are essential to prevent overfitting.
- **Robustness.** Gradient boosting can be adapted to different loss functions, allowing flexible modeling (e.g., regression, binary classification, ranking).
- **Interpretability.** Despite being an ensemble, feature importance can be extracted by aggregating across base learners.

Example. Suppose we want to predict whether a user intends to make a purchase based on two features:

- $\mathbf{X}_1$: Time spent on product page (in minutes)
- $\mathbf{X}_2$: Number of product views
- $\mathbf{Y}$: 1 if purchase intent detected, 0 otherwise

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingClassifier

# Generate synthetic data
np.random.seed(42)
time_spent = 5 + 2 * np.random.randn(200) # Minutes on page
product_views = 3 + 1.5 * np.random.randn(200) # Number of views
X = np.column_stack((time_spent, product_views))
y = ((time_spent + 2*product_views) > 11).astype(int) # Purchase intent if
    weighted sum exceeds threshold

# Fit a Gradient Boosting Classifier
gbm = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
    max_depth=2, random_state=42)
gbm.fit(X, y)

# Predict on a grid
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-1, X[:,0].max()+1, 200),
                     np.linspace(X[:,1].min()-1, X[:,1].max()+1, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
probs = gbm.predict_proba(grid)[:, 1].reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, probs, levels=np.linspace(0,1,20), cmap="RdYlBu", alpha
    =0.6)
plt.scatter(X[:,0], X[:,1], c=y, cmap="bwr", edgecolor='k')
plt.xlabel('Time␣Spent␣(Minutes)')
plt.ylabel('Product␣Views')
plt.title('Purchase␣Intent␣Prediction␣via␣Gradient␣Boosting')
plt.show()

# Feature Importance
importances = gbm.feature_importances_
print(f"Feature␣Importances:␣Time␣Spent={importances[0]:.3f},␣Product␣Views={
    importances[1]:.3f}")
```

, where Gradient Boosting sequentially corrects its previous mistakes by fitting trees to the residuals. The final decision boundary is flexible yet smooth, and feature importances reveal the relative influence of time spent and product views.

Applications. Gradient boosting models are widely applied in:

- *Purchase intent prediction:* Identifying customers likely to complete a transaction.
- *Click-through rate (CTR) estimation:* Predicting ad click probability in online advertising.
- *Fraud detection:* Detecting suspicious behaviors by sequentially correcting predictive errors.

3.4.7 Bagging: Reducing Variance via Resampling

Bagging (short for *Bootstrap Aggregating*) is an **ensemble** method that reduces model variance by training multiple versions of a base learner on different bootstrap samples and then averaging their predictions (regression) or using majority voting (classification).

Formally, bagging constructs B bootstrap datasets $\{D^{(1)}, \ldots, D^{(B)}\}$ by sampling with replacement from the training set, and fits a base learner $f^{(b)}(x)$ on each $D^{(b)}$.

Mathematical Procedure. For input x, the aggregated bagging predictor is:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} f^{(b)}(x). \tag{27}$$

In classification tasks, majority voting is used instead:

$$\hat{f}_{\text{bag}}(x) = \text{mode}\left(\{f^{(b)}(x)\}_{b=1}^{B} \right). \tag{28}$$

Theory Insights.

- **Variance Reduction.** Averaging over uncorrelated predictors reduces variance:

$$\text{Var}\left(\hat{f}_{\text{bag}}\right) = \frac{1}{B}\text{Var}(f) + \left(1 - \frac{1}{B}\right) \text{Cov}(f^{(i)}, f^{(j)}).$$

- **No Bias Correction.** Unlike boosting, bagging does not attempt to correct the bias of weak learners. It's most effective when the base learner has low bias and high variance (e.g., decision trees).
- **Stability Through Resampling.** Bagging improves unstable models by diluting the effect of individual sample fluctuations.

Example. Predicting customer satisfaction from two features:

- $\mathbf{X}_1$: Number of support tickets
- $\mathbf{X}_2$: Time since last contact (in days)
- $\mathbf{Y}$: 1 if customer is satisfied, 0 otherwise

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(42)
tickets = np.random.poisson(3, 200)
recency = np.random.normal(10, 3, 200)
X = np.column_stack((tickets, recency))
y = ((tickets < 4) & (recency < 12)).astype(int)

# Fit a Bagging Classifier with decision trees
bag = BaggingClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                        n_estimators=50, random_state=42)
bag.fit(X, y)

# Predict over grid
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-1, X[:,0].max()+1, 200),
                     np.linspace(X[:,1].min()-1, X[:,1].max()+1, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
probs = bag.predict_proba(grid)[:, 1].reshape(xx.shape)

# Plot
plt.contourf(xx, yy, probs, levels=np.linspace(0,1,20), cmap="RdYlBu", alpha
    =0.6)
plt.scatter(X[:,0], X[:,1], c=y, cmap="bwr", edgecolor='k')
plt.xlabel('Support␣Tickets')
plt.ylabel('Days␣Since␣Last␣Contact')
plt.title('Customer␣Satisfaction␣Prediction␣via␣Bagging')
plt.show()
```

, where Bagging aggregates multiple weak decision trees trained on resampled data, resulting in a smoother, more stable decision boundary. Unlike boosting, it does not focus on previously misclassified examples, making it resistant to overfitting.

Applications. Bagging is widely applied in:

- *Churn prediction:* Stabilizing noisy patterns from user histories.
- *Fraud detection:* Averaging over high-variance trees in imbalanced datasets.
- *Medical diagnostics:* Reducing overfitting in high-variance settings.

3.5 Gradient Descent

Having defined the key components of supervised learning—namely the statistical models (e.g., linear and logistic regression), the loss functions (e.g., squared error and cross-entropy), and the optimization objectives—they culminate in a common practical task: minimizing the empirical risk function over model parameters. In this context, gradient-based optimization emerges as the central engine of training modern machine learning systems.

Among various optimization techniques, **Gradient Descent (GD)** and its variants form the foundational approach for updating parameters in both classical and deep learning models. The appeal of GD lies in its simplicity, scalability, and wide applicability to differentiable loss functions. Before we delve into practical optimizers like stochastic gradient descent and adaptive methods, we first formalize the theoretical structure of GD and establish its convergence properties.

Algorithm 2 Gradient Descent

```
Initialize $\theta^{(0)}$, set $k \leftarrow 0$, set $\epsilon$ as a positive number representing the tolerance of the numerical accuracy.
while $\|\nabla \mathcal{L}(\theta^{(k)})\| \geq \epsilon$ do
    $\theta^{(k+1)} = \theta^{(k)} - t_k \nabla \mathcal{L}(\theta^{(k)})$
    $k \leftarrow k + 1$
end while
return $\hat{\theta} = \theta^{(k)}$
```

The pseudocode of the GD is shown in the Algorithm 2. The crux of the algorithm is to iteratively update the parameters via the information of the gradient at the current step. To illustrate the process of how the algorithm works, we introduce two concrete examples in the following subsection 3.5.1.

3.5.1 Examples of GD

In the subsection, suppose that we are given n data observations: $(x_i, y_i), i = 1, 2, ..., n$, where $(y_i)_{i=1}^n$ are labels. Let $X = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$.

Example 1: Estimating the OLS estimator by Gradient Descent Now we consider how to use the GD to estimate the OLS estimator. Although the OLS estimator enjoys a closed form solution $\hat{\beta} = (X^\top X)^{-1}X^\top y$, the direct computation of it suffers from the problem of scalability and numerical stability due to the inverse operation.

GD doesn't require the inverse of the matrix. Instead, it utilizes the information of the loss function. Remember that the loss function of the OLS is defined as $\mathcal{L}(\beta) = \sum_{i=1}^n |y_i - \sum_{j=1}^p x_{ij}\beta_j|^2 = ||y - X\beta||^2$, and therefore, the gradient $\nabla\mathcal{L}(\beta)$ will be $X^T(X\beta^k - y)$. And the third step in the Algorithm 2 will be $\beta^{k+1} = \beta^k - t_k X^T(X\beta^k - y)$.

Example 2: Logistic Regression by Gradient Descent Now, we turn to the logistic regression. Recall that the Logistic regression it links y_i and x_i as the conditional probability of a binary outcome given the feature. The loss function for the Logistic Regression is known as the Cross-Entropy loss, whose empirical version is defined to be $\sum_i^n \left\{(y_i \ln(\frac{\exp(x_i'\beta)}{1+\exp(x_i'\beta)}) + (1-y_i)\ln(\frac{1}{1+\exp(x_i'\beta)})\right\}$. Its gradient $\nabla\mathcal{L}(\beta)$ is given by $\frac{1}{n}\sum_{i=1}^n \left(\sigma(\beta^T x_i) - y_i\right)x_i$, where $\sigma(\cdot) = \frac{\exp(\cdot)}{1+\exp(\cdot)}$.

General Form We have grasped some sense of how the GD works for specific examples, now we turn to the summary of the workflow for the general problem.

To begin with, we define the cost function $J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^m [h_\Theta(x_i) - y_i]^2$ in terms of a hypothetical function h_Θ from some parameterized space and the label $y_i, i = 1, 2, ..., n$.

Next, we derive the iterative formula, $\Theta_j = \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J(\Theta_0, \Theta_1), j \in \{0, 1\}$, where $\frac{\partial}{\partial\Theta}J_\Theta = \frac{\partial}{\partial\Theta}\frac{1}{2m}\sum_{i=1}^m [h_\Theta(x_i) - y]^2$.

Therefore, $\Theta_j := \Theta_j - \frac{\alpha}{n}\sum_{i=1}^n [(h_\Theta(x_i) - y)x_i], j \in \{0, 1\}$.

Statistics convergence vs. Optimization convergence From now on, we have discussed a lot about how the GD works. But will GD return the estimate that we wants and how? Specifically, we are asking the convergence of the estimator, that is,

- Convergence of estimator: Is the estimator constructed by the loss function converging to the true underlying estimand? How fast? Bias and consistency?

The question has its own ”dual question” in the language of optimization, that is,

- Convergence of optimization: Will the optimization algorithm we use converge to the minimizer of the loss function given data? How fast?

3.5.2 *Gradient Descent Convergence*

In the subsection[2], we will introduce the convergence conditions for the GD. The sketch of the proof is as follows,

[2] For more details, see https://www.stat.cmu.edu/~siva/teaching/725/lec3.pdf

- The following inequality tells us when the gradient descent converges:

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L}||\nabla f(x_k)||_2^2$$

- For Lipschitz continuous functions, we can use the Taylor expansion at a point x_k to show the inequality. Then, we can use the inequality to show that, for a small enough learning rate, the function in period $k+1$ is smaller than that in period k.

- We can also leverage convexity to show the sequence decreases to the global minimum, which is by the following theorem.

  Theorem 3.1. *Suppose the function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$, i.e., we have that $||\nabla f(x) - \nabla f(y)||_2 \leq L||x-y||_2$ for any x, y. Then if we run gradient descent for k iterations with a fixed step size $t \leq 1/L$, it will yield a solution $f(x^{(k)})$ which satisfies*

$$f(x^{(k)}) - f(x^*) \leq \frac{||x^{(0)} - x^*||_2^2}{2tk}$$

  where $f(x^*)$ is the optimal value. Intuitively, this means that gradient descent is guaranteed to converge and that it converges with rate $O(1/k)$.

- The next theorem tells us the speed at which the gradient descent converges.

  Theorem 3.2. *Let $f : S \to \mathbb{R}$ be a strongly convex function with parameters m, M as in the definition above. For any $\epsilon > 0$ we have that $f(x^{(k)}) - \min_{x \in S} f(x) \leq \epsilon$ after k iterations for any k that respects:*

$$k^* \geq \frac{\log(\frac{f(x^{(0)}) - \alpha^*}{\epsilon})}{\log(\frac{1}{1-m/M})}.$$

- Basically, you need $O(1/\epsilon)$ steps to converge for convex functions. For strongly convex functions, you need $O(\log(1/\epsilon))$ steps to converge.

Besides the original framework, there are several variants of gradient descent as follows.

3.5.3 Momentum

The GD performs well if the function f enjoys great property as the Theorem 3.1 and Theorem 3.2 mention. However, in reality, the function will not always be as expected. For example, if the function is not convex, the GD usually gets trapped into the local minima; if the function is not smooth at some points(when $f := |x|$), the GD will oscillate back and forth. Hence, modified version of GD has been proposed. one of them is called the Gradient Descent with the Momentum(GDM), whose general **intuition** is as follows,

- If successive gradient steps point in different directions, we should cancel off the directions that disagree;
- If successive gradient steps point in similar directions, we should go faster in that direction.

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations, specifically, modification is done by replacing the gradient term $\nabla_\theta \mathcal{L}(\theta_k)$ with g_k, and the update rule in the Algorithm 2 is modified as $\theta_{k+1} = \theta_k - \alpha g_k$. As a comparison,

- GD: $g_k = \nabla_\theta \mathcal{L}(\theta_k)$;
- GDM: $g_k = \nabla_\theta \mathcal{L}(\theta_k) + \mu g_{k-1}$.

3.5.4 Stochastic Gradient Descent

What we have mentioned above uses all data when it calculates the gradient. However, the data set is usually large when it comes to the case of deep learning. It is computationally intractable to use all the data to update the gradient each time. Instead, we sample data to update gradient, which introduces another important version of the GD, the Stochastic Gradient Descent(SGD). Still, the modification is done by replacing the gradient term with g_k. This time, the replacement works as follows,

1. Sample $\mathcal{B} \subset \mathcal{D}$
2. Estimate $g_k \leftarrow -\nabla_\theta \frac{1}{B} \sum_{i=1}^{B} \log\ p(y_i|x_i, \theta) \approx \nabla_\theta \mathcal{L}(\theta)$
3. $\theta_{k+1} \leftarrow \theta_k - \alpha g_k$

Each iteration is called a mini-batch, wher e the batch refers to the sampling dataset $\mathcal{B}$. In practice, we shuffle data instead of randomly sampling.

3.5.5 Adaptive Gradient

The adaptive gradient has two different methods. The first method is using momentum, as we have introduced. The second method normalizes the gradient using the previous gradient in the past periods. The update rule can be seen as follows:

$$\theta^{(k)} = \theta^{(k-1)} - \alpha \cdot \frac{\nabla \mathcal{L}_k(\theta^{(k-1)})}{\sqrt{\sum_{k'=1}^{k} \nabla \mathcal{L}(\theta^{(k'-1)})^2}}.$$

3.5.6 Adam

Finally, we introduce the Adaptive Moment Estimation(Adam), rougly speaking, Adam = Momentum + Adaptive Learning Rate. It is one of the most well-used algorithms to automatically tune momentum and learning rates in practice and the default optimizer tuner in many famous deep learning computational framework like pytorch and tensorflow. The pseudocode in the Algorithm 3 explains the workflow of the Adam.

Algorithm 3 Adam

```
$M_0 = 0, R_0 = 0$
for $t = 1, \ldots, T$ do
    $M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla\mathcal{L}(\theta^{(t-1)})$ // 1st moment estimate
    $R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla\mathcal{L}(\theta^{(t-1)})^2$ // 2nd moment estimate
    $\hat{M}_t = \frac{M_t}{1-(\beta_1)^t}$ // 1st moment bias correction
    $\hat{R}_t = \frac{R_t}{1-(\beta_2)^t}$ // 2nd moment bias correction
    $\theta^{(t)} = \theta^{(t-1)} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}}$ //update
end for
return $\theta^{(T)}$
```

Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. [3] The method is straightforward to implement, computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyperparameters have intuitive interpretations and typically require little tuning.

3.6 The Notebooks

- The notebook Gradient_Descent implements the gradient descent for linear regression from scratch and compares it with the closed-form OLS estimator;

- The notebook k-Nearest_Neighbors, Random_Forest, Regression_Tree, XGBT implement the machine learning algorithms on the Boston Housing Dataset respec-

[3] For better references, seehttps://arxiv.org/pdf/1609.04747, https://www.zhihu.com/question/323747423/answer/2576604040.

tively, and specifically, they utilize the cross-validation to fine tune the hyper-parameters in the respective algorithms, i.e., the number of neighbors parameter k in the KNN regressor;

- The notebook Bootstrap simulates some data with the standard linear regression as the DGP and illustrates that bootstrap is a valid method to produce the standard errors and confidence intervals comparing with the OLS estimates.

4 Deep Neural Networks

In previous sections, we reviewed supervised learning through the lens of model definition, loss specification, and gradient-based optimization techniques. Classical models such as linear and logistic regression work well in low-dimensional settings or when the signal-to-noise structure is relatively simple. However, as real-world applications increasingly demand the modeling of complex, high-dimensional, and unstructured data—such as images, text, and speech—these classical models fall short. We now turn to a class of models that have revolutionized modern machine learning: **Deep Neural Networks (DNNs)**.

Unlike shallow models, DNNs learn expressive, hierarchical representations through multiple layers of transformation. The conceptual foundation of this approach can be traced back to the connectionist school of AI, which stands in contrast to symbolic AI.

Symbolic AI vs. Connectionist AI

- **Symbolic AI** builds systems based on logical rules, knowledge graphs, and hand-engineered representations. It dominated early AI research and powered expert systems but struggles with perception and learning from raw data.
- **Connectionist AI** mimics the structure of biological neurons and relies on data-driven learning. This paradigm includes neural networks and forms the theoretical backbone of modern deep learning.

Despite being conceptualized decades ago, neural networks long remained a niche technique due to limitations in computation, data availability, and training instability. It was not until the late 2000s that a combination of large datasets, GPU computing, and algorithmic innovations—led by pioneers such as **Geoffrey Hinton**, **Yoshua Bengio**, and **Yann LeCun**—propelled deep learning to the forefront of AI.

We do not focus on alternative nonlinear models like Support Vector Machines (SVMs) here because, although theoretically powerful in small to medium dimensions, SVMs do not scale well with very large datasets or unstructured inputs. In contrast, deep learning methods can automatically learn task-relevant representations and have shown superior empirical performance in nearly every major benchmark.

Modern AI Relies on Big Data DNNs thrive in data-rich environments. As shown in Figure 4.0.1, the amount of digital data generated annually has exploded in recent years. This explosion fuels the success of large models trained on diverse sources.

- Figure 4.0.2 demonstrates the empirical relationship between data availability and model performance across shallow, medium, and deep models. Deep architectures consistently outperform others when sufficient data are available.

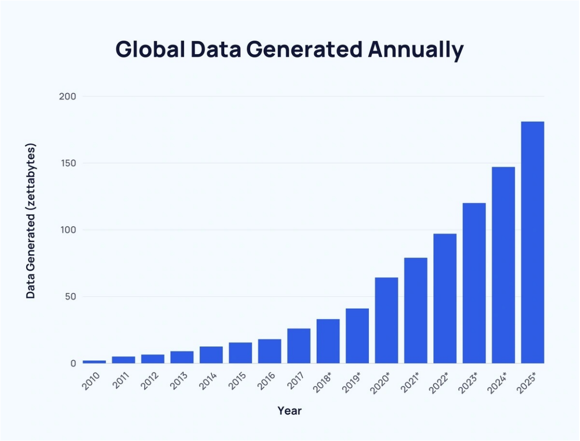


Figure 4.0.1. Annual Amount of Data

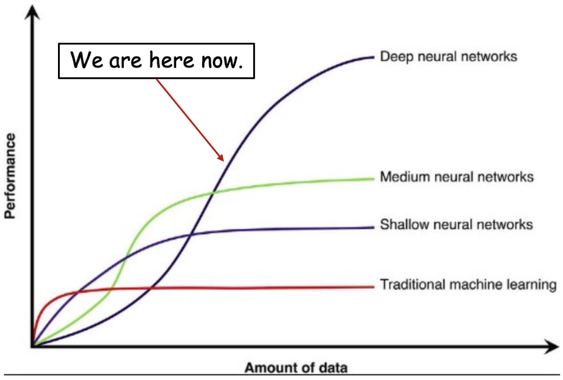


Figure 4.0.2. Relation between Performance and Amount of Data

4.1 Neural Network Structure

A deep neural network consists of a sequence of layers, each transforming the input into a new representation. The basic computational unit is the **neuron**, which computes a weighted sum followed by a non-linear activation:

$$z = \sum_i w_i x_i + b, \quad a = f(z)$$

Here, w_i are the weights, b is a bias term, and f is a non-linear activation function.

- **Input Layer**: Receives raw features (e.g., pixels, word embeddings).
- **Hidden Layers**: Extract intermediate representations through linear transformations and activations.
- **Output Layer**: Produces final predictions, such as class probabilities or regression targets.

Theorem 4.1 (Universal Approximation Theorem). [4] *A feedforward neural network with one hidden layer containing a finite number of neurons can approximate any continuous function on a compact domain, given appropriate weights and biases.*

Although this result guarantees representational power, it is non-constructive: it does not inform us how to train such a network or how many neurons are needed. In practice, deeper networks generalize better and are easier to optimize.

[4] Further details can be accessed via the wikipedia link.

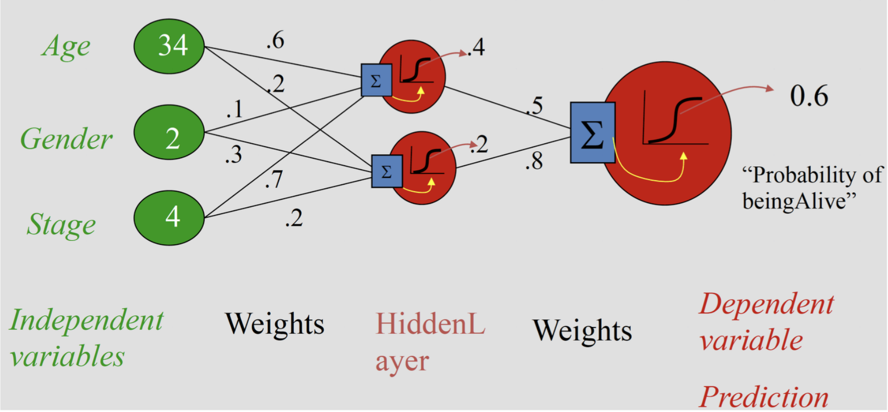


Figure 4.1.1. Architecture of a Feedforward DNN

4.2 Activation Functions

Non-linearity is introduced via activation functions:

- **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$; widely used for its simplicity and ability to avoid vanishing gradients.
- **Sigmoid**: $f(z) = \frac{1}{1+e^{-z}}$; historically popular but suffers from saturation.
- **Tanh**: $f(z) = \tanh(z)$; outputs in $[-1, 1]$ and centers activations.

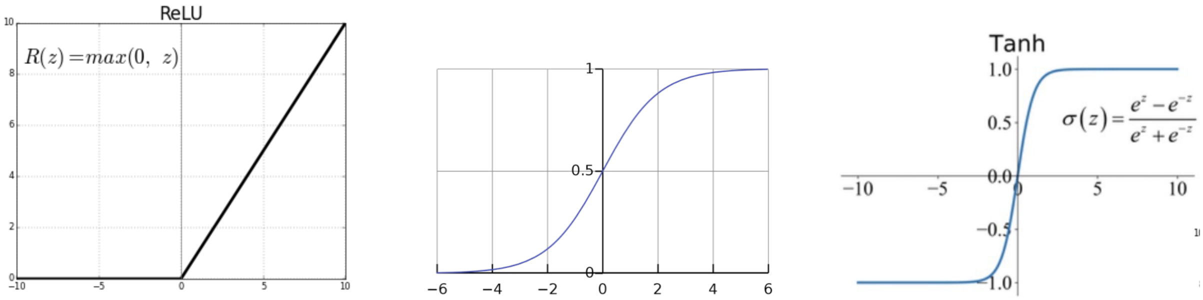


Figure 4.2.1. Sigmoid, Tanh, and ReLU Activation Functions

4.3 Training Deep Neural Networks

Training deep neural networks involves a combination of architectural design, gradient-based optimization, and careful parameter tuning. Unlike shallow models, DNNs require navigating high-dimensional, non-convex loss landscapes while coordinating the interaction between multiple layers of transformations. This section outlines the core components of training, including the forward-backward algorithm, common layer types, gradient estimation, and challenges in optimization.

4.3.1 Training Pipeline

The training process proceeds in three main stages:

1. **Forward Pass**: Input data is propagated through the network to compute predicted outputs $\hat{y} = f(x;\theta)$ and the loss $\mathcal{L}(y,\hat{y})$.
2. **Backward Pass**: Using the chain rule, gradients of the loss with respect to each parameter are computed. This procedure, known as **backpropagation**, efficiently propagates error derivatives from the output layer to all earlier layers.
3. **Parameter Update**: Model parameters are updated using gradient-based optimizers such as stochastic gradient descent (SGD) or Adam. These updates aim to minimize the training loss iteratively.

4.3.2 Common Layer Types

Deep networks are composed of different types of layers, each suited to particular input structures or tasks:

- **Fully Connected (Dense) Layers**: Every neuron in one layer connects to all neurons in the next. These layers are general-purpose and commonly used in the final stages of classification tasks.
- **Convolutional Layers**: Primarily used in computer vision, these layers apply learned filters to local regions of the input (e.g., image patches), capturing spatial hierarchies.
- **Recurrent Layers**: Useful for sequential data (e.g., language or time series), recurrent layers maintain memory over past inputs using mechanisms like RNNs, LSTMs, or GRUs.

4.3.3 Gradient-Based Estimation and Backpropagation

The core of neural network training lies in computing gradients efficiently. For instance, consider the loss function for binary classification with sigmoid output:

$$J(y,\hat{y}) = y\log\hat{y} + (1-y)\log(1-\hat{y}).$$

Using the chain rule, we compute the gradient of J with respect to a weight θ_j as:

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial \theta_j},$$

where $a = \sum_j \theta_j x_j$ is the linear pre-activation. Since $\hat{y} = \sigma(a)$ and $\frac{dJ}{da} = \hat{y} - y$, we obtain:

$$\frac{\partial J}{\partial \theta_j} = (\hat{y} - y)x_j.$$

This shows that gradient descent encourages weights to shift in the direction that reduces the prediction error.

4.3.4 Loss Landscapes and Optimization Challenges

Unlike convex optimization problems, the loss surfaces in deep networks are non-convex and can contain:

- Multiple local minima
- Saddle points with zero gradient but non-optimal curvature
- Flat regions that slow convergence

Yet, in practice, gradient-based methods often find solutions that generalize well. This empirical success can be attributed to several techniques:

- **Initialization**: Using schemes like He or Xavier initialization ensures that activations and gradients maintain reasonable variance across layers.
- **Normalization**: Batch Normalization standardizes activations within mini-batches, improving training stability and allowing higher learning rates.
- **Adaptive Optimization**: Algorithms like Adam adjust the learning rate for each parameter individually, accelerating convergence in ill-conditioned landscapes.

Overall, the combination of architectural design, forward-backward computation, and modern optimization heuristics has made deep learning tractable and scalable across diverse domains.

4.4 Overfitting and Regularization

One of the central challenges in training deep neural networks is managing the trade-off between model complexity and generalization performance. Deep models, by design, possess a vast number of parameters and are capable of fitting highly complex patterns in the training data. While this expressive capacity is crucial for tasks such as image classification or natural language understanding, it also means that neural networks can easily overfit the training set—memorizing noise or irrelevant patterns instead of learning generalizable structure.

Overfitting occurs when a model achieves low training error but performs poorly on unseen data. This is typically due to high variance in the model: it adapts too closely to the idiosyncrasies of the training data. In statistical terms, this reflects an imbalance in the bias-variance tradeoff. To mitigate overfitting, a variety of **regularization** techniques are used. These methods constrain the learning process, reduce model variance, and encourage simpler or smoother solutions.

4.4.1 Weight Penalty Methods: L1 and L2 Regularization

The most classic form of regularization penalizes the magnitude of model parameters.

- **L2 Regularization (Ridge)** adds a squared norm penalty to the loss function:

  $$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \sum_j \theta_j^2$$

  This discourages large weights by spreading influence across many small parameters. Geometrically, L2 prefers smooth solutions and helps prevent extreme fluctuations in activations.

- **L1 Regularization (Lasso)** imposes a sparsity-inducing penalty:

  $$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \sum_j |\theta_j|$$

  Unlike L2, L1 encourages many weights to become exactly zero, which can be useful for feature selection or compressing models.

- **Elastic Net** combines L1 and L2 penalties, balancing sparsity and smoothness:

  $$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda_1 \sum_j |\theta_j| + \lambda_2 \sum_j \theta_j^2,$$

  or, in the more common parametrization:

  $$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \left[\alpha \sum_j |\theta_j| + (1 - \alpha) \sum_j \theta_j^2 \right],$$

  where $\lambda \geq 0$ is the overall regularization strength and $\alpha \in [0, 1]$ controls the trade-off between L1 and L2 penalties.

  - When $\alpha = 1$, Elastic Net reduces to Lasso (pure L1).
  - When $\alpha = 0$, it becomes Ridge regression (pure L2).

These penalties can be interpreted as imposing Gaussian (L2) or Laplace (L1) priors on parameters in a Bayesian framework.

4.4.2 Dropout: Randomized Regularization

Dropout is a simple but effective technique proposed by Srivastava et al. (2014). During training, each neuron is independently "dropped" (i.e., temporarily set to zero) with probability $1 - p$:

$$h'_i = \begin{cases} h_i & \text{with probability } p, \\ 0 & \text{with probability } 1 - p. \end{cases}$$

This prevents neurons from co-adapting and forces the network to learn redundant, distributed representations. At test time, all neurons are used, but their outputs are scaled by the dropout probability p to maintain consistent expectations.

Dropout can be viewed as training an ensemble of subnetworks and averaging their predictions, leading to robust generalization.

4.4.3 Early Stopping: Validation-Guided Training

Early stopping is a regularization technique that monitors model performance on a held-out validation set during training. When the validation error stops improving (and may begin to increase), training is halted—even if the training error continues to decrease.

Formally, if $\mathcal{L}_{\text{val}}^{(t)}$ denotes validation loss at epoch t, we define a patience threshold T and stop if:

$$\mathcal{L}_{\text{val}}^{(t)} > \min_{s<t} \mathcal{L}_{\text{val}}^{(s)} + \epsilon \quad \text{for } T \text{ consecutive steps.}$$

This technique prevents the model from overfitting the training data while maintaining simplicity in implementation. It is especially useful when computational cost prohibits tuning explicit regularization hyperparameters.

4.4.4 Data Augmentation

Though not always labeled as "regularization" in theory, **data augmentation** effectively increases the training data size and diversity. For example, in image tasks, this may involve:

- Horizontal/vertical flipping
- Random cropping and rotation
- Color jittering or brightness adjustment

By exposing the model to slightly varied versions of each input, data augmentation promotes invariance and reduces reliance on exact patterns in the training set.

4.4.5 Batch Normalization and Implicit Regularization

Batch Normalization (BatchNorm) standardizes layer inputs during training, stabilizing gradient flow. While not originally designed as a regularizer, BatchNorm often reduces overfitting due to its smoothing effect on the optimization landscape.

Furthermore, recent work suggests that SGD itself introduces an **implicit regularization** bias toward flatter minima—solutions with low curvature and better generalization—particularly in overparameterized networks.

4.4.6 Summary

Table 7. Comparison of Regularization Techniques

Method	Effect
L2 (Ridge)	Shrinks weights smoothly
L1 (Lasso)	Induces sparsity in weights
Dropout	Prevents co-adaptation via random deactivation
Early Stopping	Stops training before overfitting
Data Augmentation	Expands dataset diversity
BatchNorm	Stabilizes and smooths training

4.5 Implementation with DL Libraries

The practical success of modern deep learning owes much to the powerful and evolving ecosystem of open-source software libraries. These frameworks abstract away many low-level details—such as tensor algebra, gradient computation, and hardware acceleration—allowing researchers and practitioners to rapidly prototype, train, and deploy large-scale models.

4.5.1 Core Deep Learning Frameworks

- **PyTorch**: Developed by Meta AI (Facebook), PyTorch is a dynamic computation graph library. Its key strength lies in its intuitive and Pythonic interface, which closely resembles NumPy. PyTorch supports imperative programming (eager execution), making it easy to debug and experiment with new model structures. It has become the default framework for academic research and increasingly for production use.

- **TensorFlow**: Developed by Google Brain, TensorFlow provides a more static and graph-based computation model. While early versions required defining full computation graphs before execution, recent versions (with `tf.function` and eager mode) have improved flexibility. TensorFlow is widely adopted in industry, especially when scalability, cross-platform deployment (e.g., TensorFlow Lite for mobile), and production pipelines are priorities.

Both frameworks offer:

- **Automatic Differentiation**: Users define the forward computation, and the framework computes gradients via backpropagation (reverse-mode autodiff).
- **GPU Acceleration**: Seamless usage of CUDA-enabled GPUs for high-performance matrix and tensor operations.
- **Integration with CUDA, cuDNN, and Tensor Cores**: Leveraging low-level libraries for optimized performance.
- **Serialization and Deployment APIs**: Support for saving models and exporting them for serving (e.g., ONNX, TensorFlow Serving, TorchScript).

4.5.2 High-Level Abstractions and Model Hubs

- **Keras**: Originally an independent project, now officially integrated with TensorFlow as `tf.keras`. Keras provides a high-level API for defining, training, and evaluating deep learning models. It is especially beginner-friendly, with minimal boilerplate code required to build complex architectures.
- **HuggingFace Transformers**: A modern library for natural language processing that provides pre-trained transformer-based models (e.g., BERT, GPT, T5). It offers a unified interface to load, fine-tune, and deploy state-of-the-art models with minimal code. HuggingFace also supports both PyTorch and TensorFlow backends.

These abstractions enable rapid experimentation and fine-tuning through pre-built modules, pretrained checkpoints, and standard training loops (via `Trainer`, `compile()`, etc.).

4.5.3 Practical Benefits of Using DL Libraries

- **Efficiency**: Optimized linear algebra routines (e.g., matrix multiplication, convolution) implemented in C++/CUDA and exposed via Python.

- **Modularity and Reusability**: Model components (layers, optimizers, losses) can be composed as reusable modules, aiding scalability and maintainability.
- **Experiment Tracking and Logging**: Integration with tools like TensorBoard, Weights & Biases, and MLflow allows real-time monitoring of training progress, losses, and hyperparameters.
- **Reproducibility**: Built-in random seed control and deterministic computation modes help ensure experimental consistency.
- **Deployment**: Support for exporting models to various formats (e.g., TorchScript, ONNX, SavedModel) facilitates serving on cloud, mobile, and edge devices.

4.5.4 Which Library to Choose?

- Use `PyTorch` if you prioritize flexibility, readable code, and cutting-edge research integration (most new papers are prototyped in PyTorch).
- Use `TensorFlow/Keras` if your workflow includes large-scale industrial deployment or mobile/embedded integration.
- Use `HuggingFace Transformers` if you're working on NLP tasks and need access to pretrained transformer models with strong community support.

4.6 The Notebooks

- The notebook Dropout implements the dropout technique, one of the famous regularization technique in the neural network;
- The notebook He_initialization implements the initialization technique proposed by Kaiming He;
- The notebook Chain_Rule implements the chain rule operation for obtaining the gradient of a logistic regression model;
- The notebook Micrograd implements a simpler but elegant gradient symbolic computation framework, modfied from the one implemented by Andrej Karpathy. The framework enables us to obtain the gradient by imputing the mathematical expression.

5 Computations in Deep Learning

Training deep neural networks is not only a statistical and algorithmic challenge, but also a computationally intensive endeavor. As model architectures grow in depth, width, and complexity, the demand for specialized hardware and scalable infrastructure becomes a dominant concern. In this section, we discuss the practical aspects of deep learning computation, covering hardware options, GPU performance, model training times, and the financial costs of training large-scale models.

5.1 Hardware Platforms for Deep Learning

Modern deep learning workflows typically rely on two classes of computing platforms:

5.1.1 Local Workstations and Servers

Enthusiasts, research groups, and smaller labs often build self-hosted machines with top-tier consumer GPUs. A common setup includes:

- A high-performance CPU with an **NVIDIA RTX 4090** GPU (market value: ∼US$1,500, though often hard to obtain due to demand).
- For institutional setups, such as the CUHK Business School's DOT department, server-grade infrastructure is deployed. For example:
  - Two servers with 16 RTX 4090s are already active.
  - Two to three additional servers, each equipped with 16–24 NVIDIA H100 GPUs, are planned for deployment.

5.1.2 Cloud Infrastructure

Large-scale training tasks often rely on cloud computing due to scalability and convenience. Key platforms include:

- **Google Cloud Platform (GCP)**: Approx. US$38.84/hour for 8 H100 GPUs.
- **Amazon Web Services (AWS)**: Approx. US$39.33/hour for similar compute setups.

These platforms support distributed training frameworks and enable rapid scaling across nodes.

5.2 GPU Benchmarking and Comparison

The choice of GPU has a profound impact on training time and throughput. Figure 5.2.1 presents benchmark comparisons across popular GPUs such as the NVIDIA A100, H100, and RTX 4090.

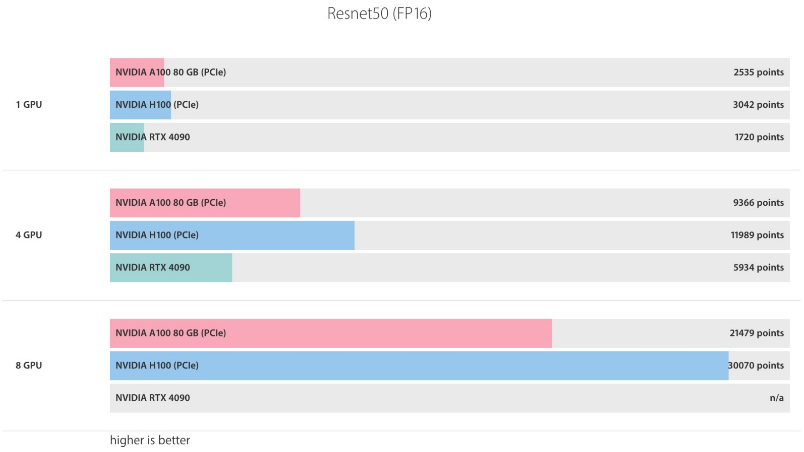


Figure 5.2.1. Comparison of GPU Performance: A100 vs. H100 vs. RTX 4090

H100s significantly outperform 4090s on both FP32 and tensor core operations. However, 4090s offer a cost-effective solution for researchers constrained by budget.

5.3 Model Size and Training Time Estimates

The training time for a neural network depends on three factors:

1. Model size (number of parameters),
2. Dataset size,
3. Compute throughput (FLOPs per second).

We provide several concrete examples:

- **ResNet-50** (12M parameters on ImageNet):
  - ∼30 minutes on a DGX server with 8 A100s.
  - Estimated ∼6–10 hours on a workstation with 1–2 RTX 4090s.
- **BERT Base** (110M parameters on BooksCorpus + Wikipedia):
  - Pretraining takes roughly 5 hours on a DGX system.
  - Fine-tuning on a small dataset (e.g., SQuAD) takes only 3–5 minutes.
- **GPT-3** (175B parameters, 300B tokens):

  - Estimated compute: 3.15×10^{23} FLOPs.
  - Compute throughput: 128 DGX servers (each with 8 A100s at 80 TFLOP/s).
  - Estimated training duration: $\sim$51 to 100 days, depending on parallel efficiency.

These estimates highlight the exponential growth of resource requirements with model size.

5.4 Case Study: Compute Costs of DeepSeek-V3

Recent advancements in **mixture-of-expert** (MoE) architectures such as DeepSeek-V3 demonstrate the economics of large model training. DeepSeek-V3 is a 671B parameter model trained on 14.8 trillion tokens.

- Each token activates only 37B parameters, reducing active compute per step.
- Total compute: $14.8 \times 10^{12} \times 37 \times 10^{9} \times 6 = 3.3 \times 10^{24}$ FLOPs.
- Cost-efficient training was achieved with H800 GPUs, yielding:

$$\text{Peak throughput} = \frac{3.3 \times 10^{24}}{9.6 \times 10^{9}\ \text{GPU-seconds}} \approx 3.4 \times 10^{14}\ \text{FLOPs/sec.}$$

Training costs are summarized in Table 8.

Training Costs	Pre-Training	Context Extension	Post-Training	Total
H800 GPU Hours	2.664M	119K	5K	2.788M
Cost (USD)	$5.328M	$0.238M	$0.01M	$5.576M

Table 8. Estimated Cost Breakdown for DeepSeek-V3

5.5 Geopolitical Constraints: GPU Export Bans

The increasing strategic importance of advanced AI compute has led to export controls on high-end GPUs. Restrictions imposed by the U.S. government in 2023–2024 ban the sale of A100, H100, and other high-performance GPUs to certain countries.

Such constraints have led to the development of downgraded variants (e.g., A800, H800) for restricted markets, and incentivized domestic semiconductor investment in affected regions.

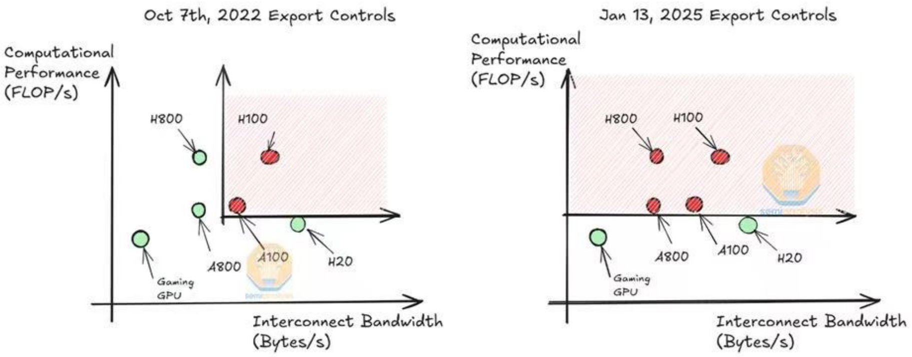


Figure 5.5.1. Overview of U.S. GPU Export Controls

Deep learning is as much a computational discipline as it is a statistical one. Advances in model architectures must be accompanied by parallel advancements in hardware efficiency and software optimization. As the scale of models continues to grow, the financial and infrastructural burden of training them will remain a core issue for researchers, institutions, and governments alike.

Chapter 3: Large Language Models

Prewords

So far, we have focused on the theoretical foundations of deep learning, including the architecture and training of deep neural networks. In this section, we apply those principles to a domain where deep learning has had especially transformative effects: **Natural Language Processing (NLP)**.

NLP aims to equip machines with the ability to understand, generate, and interact with human language. While early approaches relied on hand-crafted rules or symbolic logic, modern NLP is driven by large-scale neural models and massive corpora. The progression from symbolic systems to data-driven, transformer-based architectures illustrates the broader shift in AI from reasoning to representation learning.

The development of NLP can be understood in four distinctive stages:

Stage I: Probability and Vector Representations (Pre-2010)

Early NLP approaches focused on shallow statistical models and symbolic rules. Language was represented using handcrafted features or sparse vectors such as Bag-of-Words and TF-IDF. Probabilistic models like Hidden Markov Models (HMMs) and n-gram language models were employed for tasks such as speech recognition and part-of-speech tagging (Jurafsky and Martin, 2009).

These methods, though mathematically elegant, suffered from data sparsity and limited generalization.

Stage II: Word Embeddings and Recurrent Networks (2013–2017)

The second wave of NLP innovation was driven by the advent of distributed representations and deep sequence models. Word embeddings such as **Word2Vec** (Mikolov et al., 2013) and **GloVe** (Pennington et al., 2014) allowed words to be represented as dense vectors in a semantic space, capturing syntactic and semantic relationships through vector arithmetic.

Simultaneously, Recurrent Neural Networks (RNNs) and their gated variants—**Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRU)**—enabled neural networks to model sequential dependencies in text. This architecture laid the groundwork for end-to-end learning in tasks such as speech recognition and sentiment analysis.

A landmark development in this period was the introduction of the **Sequence-to-Sequence (Seq2Seq)** model by Sutskever et al. (2014), which used an LSTM encoder-decoder framework for neural machine translation. The model demonstrated that it was possible to translate variable-length input sequences to variable-length output sequences entirely within a neural architecture, significantly outperforming traditional statistical methods at the time.

Stage III: Transformers and Pretrained Language Models (2018–2022)

The introduction of the transformer architecture by Vaswani et al. (2017) marked a paradigm shift. By replacing recurrence with self-attention, transformers achieved both superior accuracy and computational scalability.

This enabled large-scale pretraining, as exemplified by:

- **BERT** (Devlin et al., 2018): A deep bidirectional encoder pretrained using masked language modeling.
- **GPT series** (Radford et al., 2018a, 2019; Brown et al., 2020b): Unidirectional transformers trained for autoregressive language modeling.
- **T5, RoBERTa, XLNet**(Raffel et al., 2020; Liu et al., 2019; Yang et al., 2019), and others that extended or refined the transformer framework.

Stage IV: Large Language Models and Instruction Tuning (2023–Present)

The current era is defined by the emergence of large language models (LLMs) such as **GPT-4**, **Claude**, **Gemini**, and **DeepSeek-V3**. These models are characterized by:

- **Instruction tuning** and **RLHF** to align model outputs with human intent.
- **In-context learning**, enabling few-shot or zero-shot generalization without gradient updates.
- **Scalability**: Hundreds of billions of parameters trained on trillions of tokens.

LLMs are capable not only of syntactic fluency but also of semantic reasoning and multi-turn interaction—blurring the lines between language modeling and general AI.

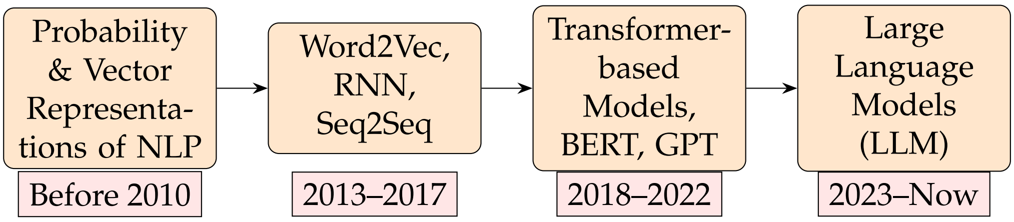

Probability & Vector Representations of NLP
Before 2010
Word2Vec, RNN, Seq2Seq
2013–2017
Transformer-based Models, BERT, GPT
2018–2022
Large Language Models (LLM)
2023–Now

Figure 5.5.2. Roadmap of NLP Development

6 From Machine Translation to Transformers: A Genealogy

Natural Language Processing (NLP) has historically treated **machine translation (MT)** as one of its most prominent and ambitious tasks. From rule-based systems to statistical models, and eventually to deep neural architectures, MT has been both a proving ground for theoretical innovation and a driver of practical breakthroughs. The emergence of attention mechanisms and transformers was not sudden, but rather the culmination of a lineage of ideas. In this section, we trace that evolution in several stages, drawing on both algorithmic principles and cognitive metaphors.

6.1 Neural Machine Translation (NMT)

Neural Machine Translation (NMT) represents a significant milestone in the evolution of machine translation systems. Unlike traditional approaches, which rely on separate components for modeling translation rules, word alignments, and fluency (e.g., phrase tables, alignment models, and n-gram language models), NMT reframes the entire problem as a single learning task: mapping an input sequence in the source language to an output sequence in the target language using a deep neural network.

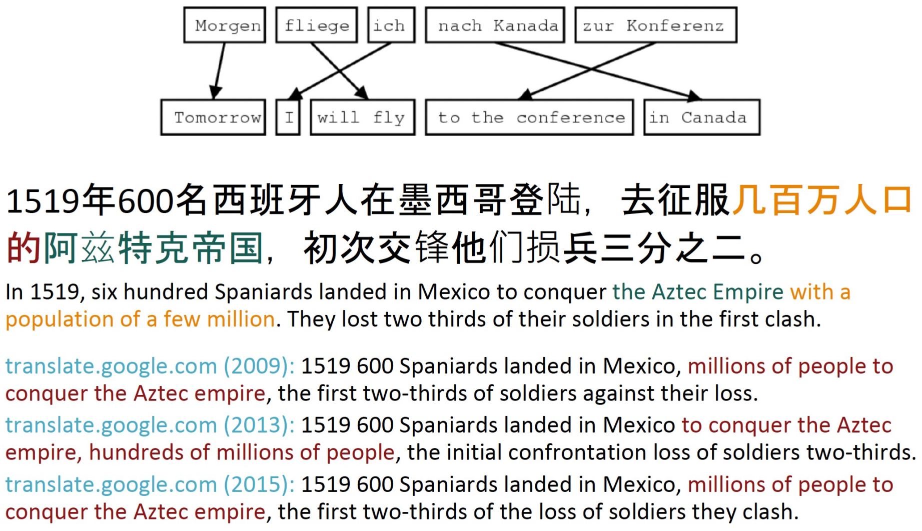
Formally, NMT aims to model the conditional probability of a target sentence $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ given a source sentence $\mathbf{x} = (x_1, x_2, \ldots, x_S)$. This is typically achieved by training a neural network to minimize the cross-entropy loss between predicted tokens and ground truth tokens.

A key advantage of NMT is that it is trained end-to-end: the entire system is optimized jointly via gradient descent, without the need for manually designed features or heuristic alignment strategies. This unified framework has enabled rapid progress in translation quality, scalability, and language generalization.

Yet, the performance of NMT hinges critically on how the source and target sequences are represented and transformed. The awkwardness of Chinese-English translation seen in the figure 6.1.1 given an intricate Spanish invasion of Mesoamerica context: This leads to the architectural design that dominated early NMT research: the sequence-to-sequence (Seq2Seq) model, which we examine next.

6.2 Sequence-to-Sequence Architecture: The Encoder-Decoder Model

Sequence-to-Sequence (Seq2Seq) modeling is a foundational neural architecture for machine translation and other sequence transduction tasks. It enables learning mappings from variable-length input sequences to variable-length output sequences by using a pair of recurrent networks: an encoder and a decoder, as can be seen in the



1519年600名西班牙人在墨西哥登陆，去征服几百万人口的阿兹特克帝国，初次交锋他们损兵三分之二。

In 1519, six hundred Spaniards landed in Mexico to conquer the Aztec Empire with a population of a few million. They lost two thirds of their soldiers in the first clash.

translate.google.com (2009): 1519 600 Spaniards landed in Mexico, millions of people to conquer the Aztec empire, the first two-thirds of soldiers against their loss.

translate.google.com (2013): 1519 600 Spaniards landed in Mexico to conquer the Aztec empire, hundreds of millions of people, the initial confrontation loss of soldiers two-thirds.

translate.google.com (2015): 1519 600 Spaniards landed in Mexico, millions of people to conquer the Aztec empire, the first two-thirds of the loss of soldiers they clash.

Figure 6.1.1. The Outcry of Cortés

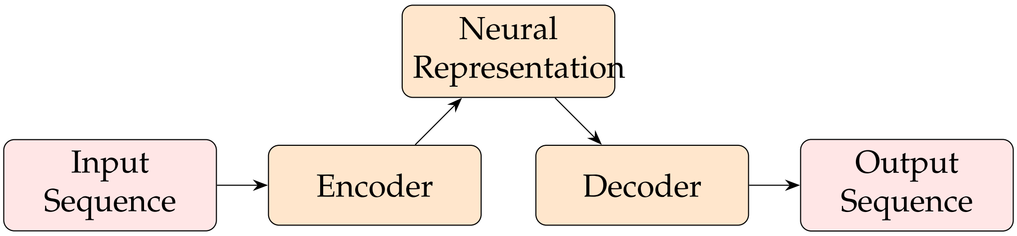


Figure 6.2.1. Sequence-to-Sequence modeling

figure 6.2.1.

The Seq2Seq paradigm was first introduced by Sutskever et al. (2014) and has since become a cornerstone of early neural machine translation (NMT) systems.

At its heart, Seq2Seq operates as a **conditional language model**. Given a source sentence $\mathbf{x} = (x_1, \ldots, x_S)$, the model generates a target sentence $\mathbf{y} = (y_1, \ldots, y_T)$ by modeling the conditional probability:

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, \mathbf{x})$$

This formulation assumes that each target token y_t is generated conditioned on all previously generated tokens $y_{<t}$ and the full source sentence $\mathbf{x}$.

6.2.1 The Architecture

The model decomposes into two main components:

Encoder RNN. The encoder processes the input sequence one token at a time, maintaining a hidden state that evolves according to a recurrent function (typically an

LSTM or GRU):

$$h_t = f(h_{t-1}, x_t)$$

After consuming the final input token x_S, the encoder outputs a final hidden state h_S, known as the **context vector**. This vector is intended to summarize the entire input sequence and is passed to the decoder as the initial condition for generation.

Decoder RNN. The decoder is also a recurrent network, initialized with the context vector h_S. At each decoding step, it uses the previously generated token y_{t-1} and its own hidden state s_{t-1} to produce a new word:

$$s_t = g(s_{t-1}, y_{t-1})$$

This state is used to generate a probability distribution over the output vocabulary for the next word. The process begins with a special `<START>` token and terminates upon generating an `<END>` token.

The overall Seq2Seq architecture is illustrated below using a staggered layout, which reflects the flow of computation from input to output via a latent representation. For example, the left part of Figure 6.2.2 shows the encoder RNN, which processes the source sentence token-by-token (”il”, ”m’”, ”a”, ”entarté”) and produces a final hidden state (context vector) summarizing the entire input sequence. This context vector serves as the initial hidden state for the decoder RNN on the right side. The decoder RNN acts as a conditional language model, generating the target sentence (”he hit me with a pie”) one word at a time. During inference (test time), each generated output token is fed back into the decoder as the input for the next prediction step. This diagram clearly highlights the two key stages of Seq2Seq: encoding the source sequence into a fixed-size vector and decoding it into the output sequence conditioned on the encoding.

6.2.2 *Training of Seq2Seq*

- **Seq2Seq models** are trained end-to-end by minimizing the loss over the entire target sentence.
- The loss J is defined as the average over all time steps:

  $$J = \frac{1}{T} \sum_{t=1}^{T} J_t$$

  where J_t is the negative log-likelihood of predicting the correct target word at time step t.

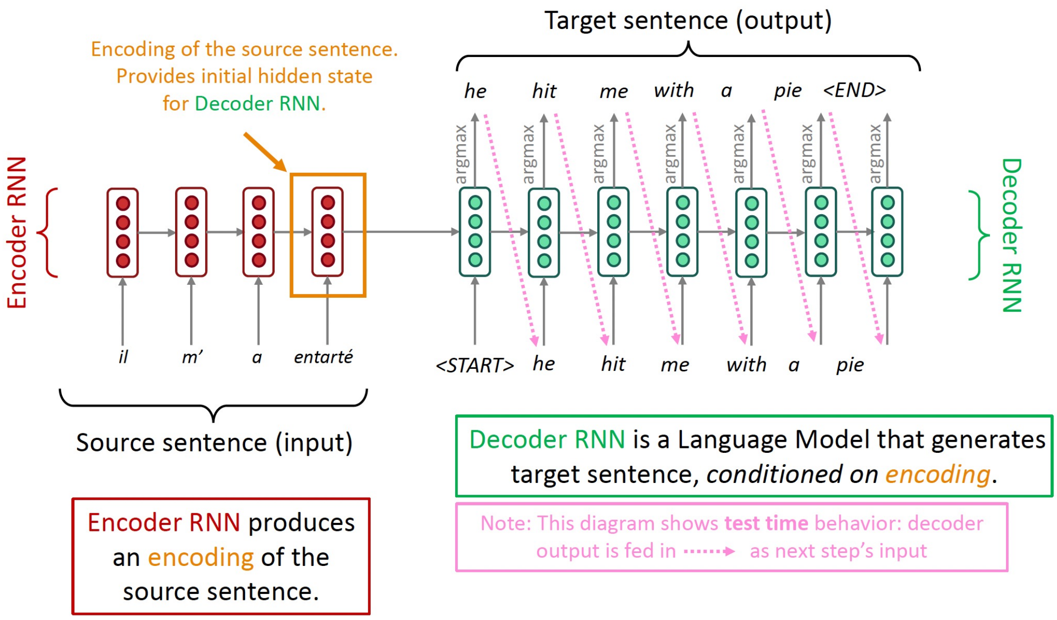

Target sentence (output)
Encoding of the source sentence. Provides initial hidden state for Decoder RNN.
he hit me with a pie <END>
argmax
Encoder RNN
Decoder RNN
il m' a entarté
<START> he hit me with a pie
Source sentence (input)
Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.
Encoder RNN produces an encoding of the source sentence.
Note: This diagram shows **test time** behavior: decoder output is fed in ······▸ as next step's input

Figure 6.2.2. Seq2Seq Architecture

- **Optimization**: Seq2Seq is optimized as a **single system**. Gradients are backpropagated through both the encoder and decoder, fully end-to-end.

6.2.3 Applications of Seq2Seq Modeling

While Seq2Seq was originally developed for machine translation, its general framework applies to a wide range of sequence transformation tasks:

- **Summarization:** Generating concise summaries from long-form documents.
- **Dialogue Systems:** Producing contextually appropriate replies in chatbots or conversational agents.
- **Parsing:** Mapping raw text into structured representations like syntax trees or logical forms.
- **Code Generation:** Translating natural language prompts into executable code (e.g., Python or SQL).

6.2.4 Limitations of Seq2Seq with RNNs

While the Sequence-to-Sequence (Seq2Seq) architecture with recurrent neural networks (RNNs) represented a major leap forward of the neural machine translation (NMT), it also suffers from several intrinsic limitations. These challenges stem from the structural and computational constraints of RNNs and ultimately restrict the model's ability to scale, generalize, and perform well on long or complex sequences.

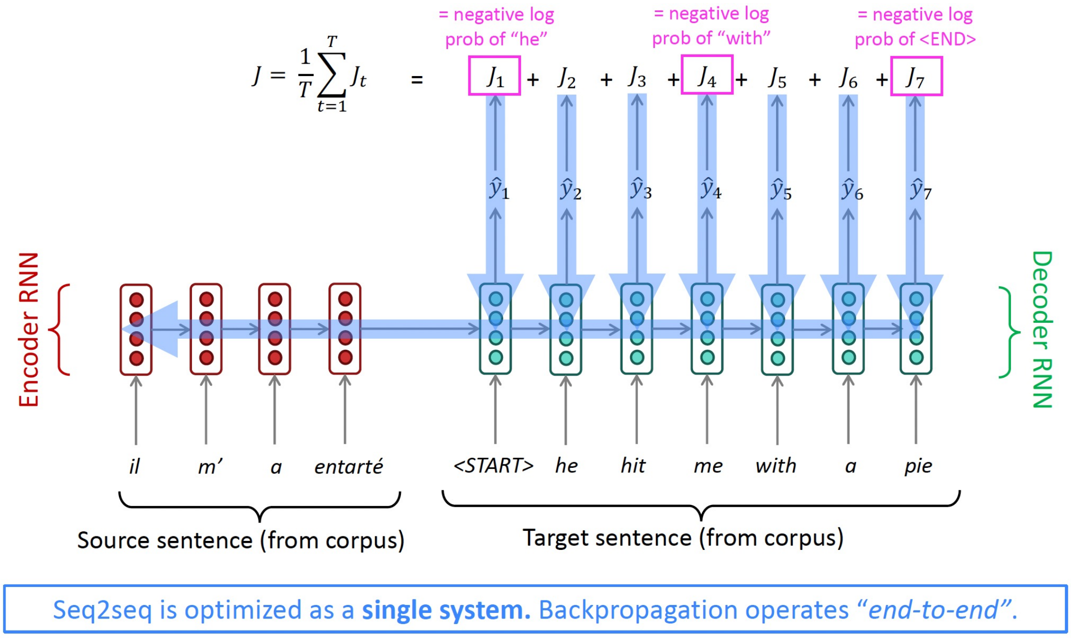

= negative log prob of "he"
= negative log prob of "with"
= negative log prob of <END>
$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$
Encoder RNN
Decoder RNN
il
m'
a
entarté
<START>
he
hit
me
with
a
pie
Source sentence (from corpus)
Target sentence (from corpus)
Seq2seq is optimized as a single system. Backpropagation operates "end-to-end".

Figure 6.2.3. Seq2Seq Training and Loss Computation

Information Bottleneck. A fundamental issue in early Seq2Seq models is the **information bottleneck** created by encoding the entire input sequence into a fixed-size context vector. In practice, this means that regardless of the input length—be it a short phrase or a multi-clause sentence—the encoder is expected to distill all relevant information into a single vector (often the final hidden state of the encoder RNN). This compression leads to the loss of detailed or long-range information, especially for longer sequences.

Important semantic cues at the beginning of a sentence can be diluted or forgotten by the time the encoder reaches the end. As a result, the decoder may struggle to accurately reproduce key content from the source sentence, especially under complex grammatical structures or morphological richness.

Linear Interaction Distance. RNNs process sequences in a strictly left-to-right (or right-to-left) order. This sequential nature means that dependencies between distant words—common in natural language—require traversing many time steps before they can interact.

In contrast to human language processing, which often leverages hierarchical and recursive structures (e.g., subject-verb agreement across clauses), RNNs impose a linear interaction constraint: a word at position 1 cannot directly interact with a word at position 50 without passing through all the intermediate steps. This makes it difficult to capture long-range dependencies or syntactic structures that span clauses or sentences.

Non-parallelizability. From a computational perspective, RNNs are inherently sequential. At each time step, the current hidden state depends on the previous hidden state, which means that neither forward passes nor backward gradients can be computed in parallel across time steps. This poses serious limitations for training on modern hardware like GPUs, which are optimized for parallel computation.

In practical terms, this means that both training and inference in RNNs require $\mathcal{O}(T)$ sequential steps, where T is the length of the sequence. As sequences grow longer, training becomes slower and harder to scale, especially in large datasets or real-time settings. Moreover, during inference, the decoder cannot generate all output tokens simultaneously but must proceed one token at a time, which is inefficient for applications like simultaneous translation or interactive dialogue.

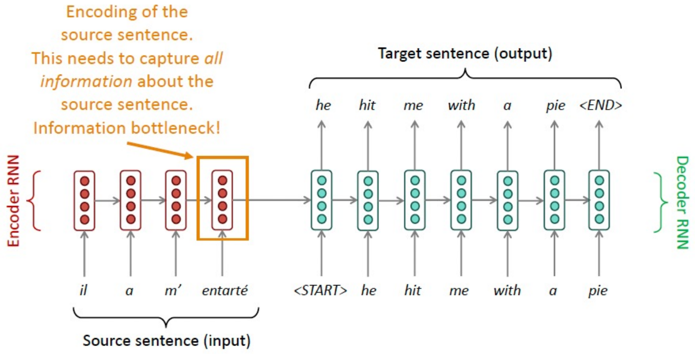


Figure 6.2.4. Information bottleneck: compression into a single context vector

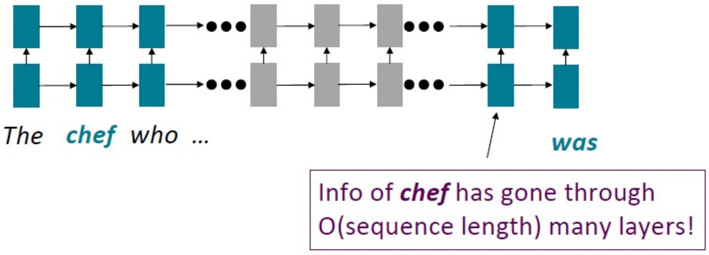

The
chef
who ...
was
Info of chef has gone through
O(sequence length) many layers!

Figure 6.2.5. Linear interaction: distant tokens weakly connected

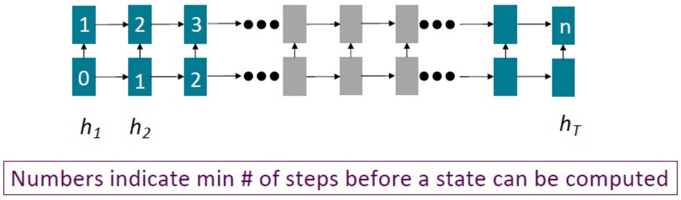

1
2
3
n
0
1
2
h_1
h_2
h_T
Numbers indicate min # of steps before a state can be computed

Figure 6.2.6. Non-parallelizability: time-step dependencies block parallel computation

These three interlocking limitations—semantic compression, interaction rigidity, and computational inefficiency—highlight the structural bottlenecks of RNN-based

Seq2Seq models. In response to these challenges, researchers introduced the **attention mechanism**, which allows the decoder to flexibly access different parts of the input sequence during generation. Rather than relying on a single context vector, attention distributes focus across all encoder hidden states, enabling both richer representations and better long-distance dependency modeling.

In the next subsection, we explore how attention models resolve these bottlenecks and sets the stage for transformer architectures.

6.3 Attention Mechanisms

The **attention mechanism** has emerged as a transformative concept in deep learning, enabling models to selectively focus on relevant parts of their input during computation. It was initially developed to address key shortcomings of Recurrent Neural Network (RNN)-based architectures—particularly their inability to efficiently model long-range dependencies and the information bottleneck caused by compressing entire input sequences into fixed-size vectors.

At a high level, attention enables a model to perform a form of content-based retrieval: it compares a **query** vector to a set of **key** vectors and retrieves associated **value** vectors in a weighted manner. Formally, given a query $q \in \mathbb{R}^d$, a set of keys $K = \{k_1, \dots, k_n\}$, and corresponding values $V = \{v_1, \dots, v_n\}$, the attention output is computed as:

$$\text{Attention}(q, K, V) = \sum_{i=1}^{n} \alpha_i v_i, \quad \text{where} \quad \alpha_i = \frac{\exp(q^\top k_i)}{\sum_{j=1}^{n} \exp(q^\top k_j)}$$

The attention weights α_i are produced by applying a softmax over the dot product between the query and each key, quantifying the similarity (or compatibility) between them. This produces a convex combination of the values v_i, weighted by their relevance to the query q. The output is thus a **context vector**—a dynamic and query-specific summary of the information contained in V.

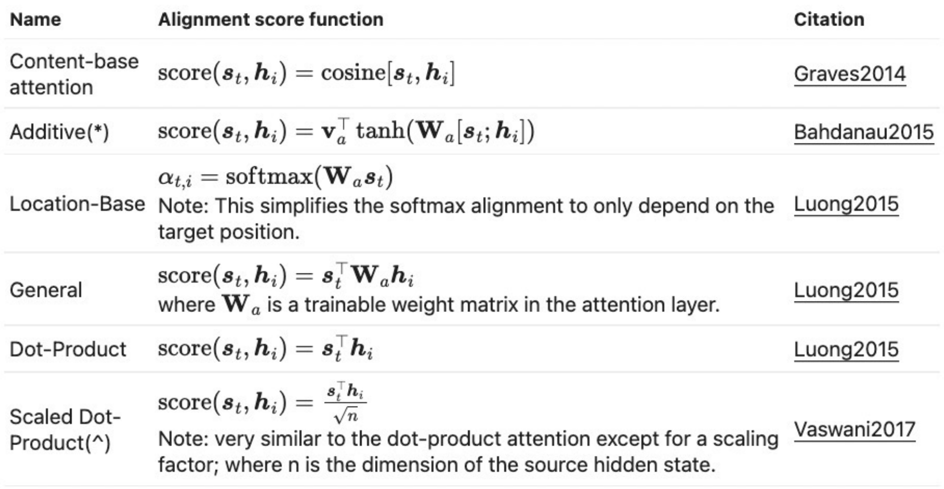
This mechanism offers several advantages:

- It removes the constraint of fixed-size encodings.
- It enables better modeling of long-distance dependencies by computing relevance globally.
- It is differentiable and fully trainable within end-to-end models.

In the context of **neural machine translation (NMT)**, attention was first introduced by Bahdanau et al. (2015). Their model augmented the Seq2Seq encoder-decoder architecture by allowing the decoder to attend to all hidden states of the encoder, instead of relying solely on the final hidden state. At each decoding time step t, the decoder generates a query vector s_t (its own hidden state), and computes attention scores against all encoder hidden states $\{h_1, \ldots, h_S\}$, which serve as both keys and values. The resulting context vector c_t is then combined with the decoder state to predict the next word:

$$c_t = \sum_{i=1}^{S} \alpha_{ti} h_i, \quad \text{where } \alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{j=1}^{S} \exp(e_{tj})}, \quad \text{and } e_{ti} = \text{score}(s_t, h_i)$$

The score function e_{ti} can take various forms (e.g., additive, dot-product, or general), but all serve the same goal: computing how relevant each source position i is to generating the current target word. This architecture allows the model to ”look back” selectively at source tokens, enabling finer-grained alignment between source and target phrases, as well as improved handling of longer and more complex input sentences:



Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Figure 6.3.1. General attention mechanism

6.3.1 A Family of Attention Models

Multiple scoring functions have been proposed for computing the alignment between queries and keys:

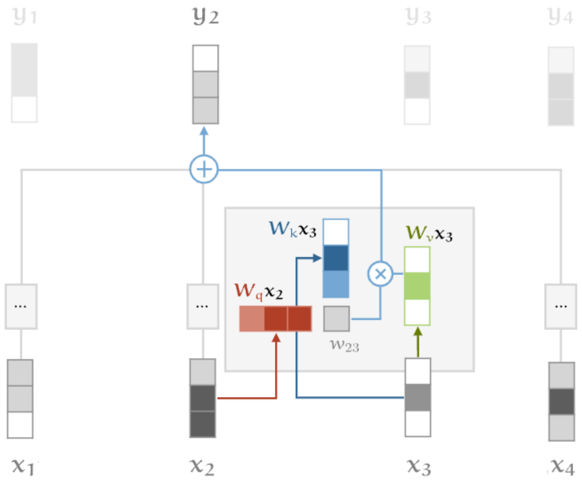


Illustration of the self-attention with key, query and value transformations.

Figure 6.3.2. Comparison of different attention mechanisms

6.3.2 *Attention is all you Need*

The **Transformer** model proposed by Vaswani et al. (2017) radically departs from RNN-based sequence modeling by removing all recurrence. Instead, it relies entirely on attention mechanisms—specifically, **self-attention**—to capture dependencies between tokens, regardless of their distance in the sequence. This architecture offers substantial advantages in parallelizability, scalability, and the ability to model long-range interactions.

The compatibility between token i and token j is measured by the scaled dot-product between the query and key vectors:

$$w'_{ij} = \frac{q_i^\top k_j}{\sqrt{d_k}}$$

where d_k is the dimensionality of the key vectors. Scaling by $\sqrt{d_k}$ prevents the dot product from becoming too large in magnitude, which can lead to vanishing gradients after applying the softmax.[5]

The attention weights are then computed via a softmax over all keys:

$$w_{ij} = \text{softmax}(w'_{ij}) = \frac{\exp(w'_{ij})}{\sum_{j'=1}^{n} \exp(w'_{ij'})}$$

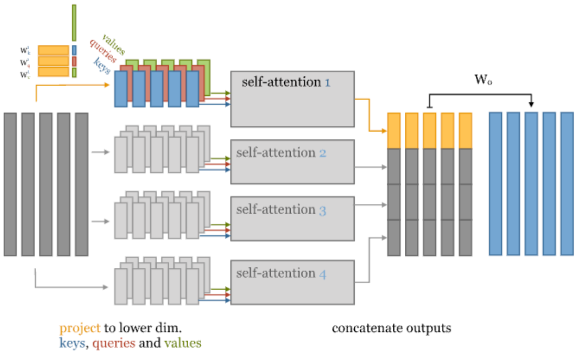
These weights determine how much token i should attend to token j. The final

[5] FAQ in NLP interview.

output for position i is the weighted sum of all value vectors:

$$y_i = \sum_{j=1}^{n} w_{ij} v_j$$

This mechanism enables each token to gather contextual information from the entire input sequence in a single step, without recursion or sequential processing. The entire self-attention computation can be efficiently parallelized across all positions.



The basic idea of multi-head self-attention with 4 heads. To get our keys, queries and values, we project the input down to vector sequences of smaller dimension.

Figure 6.3.3. Self-attention mechanism: each token attends to all others, including itself

Multi-Head Attention One of the core innovations of the Transformer architecture is the use of **multi-head attention**. Rather than computing a single attention function over the input, the Transformer applies multiple attention mechanisms—referred to as "heads"—in parallel. Each head operates in its own subspace of the input and learns to focus on different aspects of the data, such as syntax, semantics, or positional relationships.

Concretely, given an input sequence represented by matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$, multi-head attention projects this input into multiple sets of queries, keys, and values:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where:

- $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learned projection matrices for the i-th head,
- $d_k = d_{\text{model}}/H$ is the dimensionality of each head (assuming H heads).

Each attention head performs scaled dot-product attention independently. The outputs of all heads are then concatenated and linearly transformed:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)W^O$$

where $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ is another trainable weight matrix that projects the concatenated output back into the original model dimension.

This structure allows the model to:

- Attend to information from multiple representation subspaces at different positions,
- Learn a richer set of transformations compared to single-head attention,
- Parallelize the computation efficiently across all heads and positions.

For example, with $d_{\text{model}} = 256$ and $H = 8$, each head operates in a 32-dimensional subspace, and their outputs are concatenated into a 256-dimensional vector.

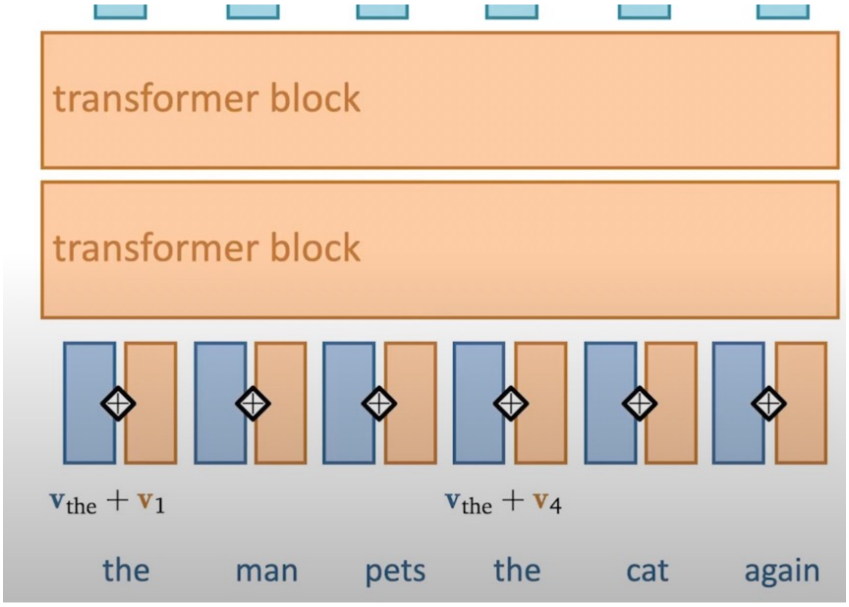
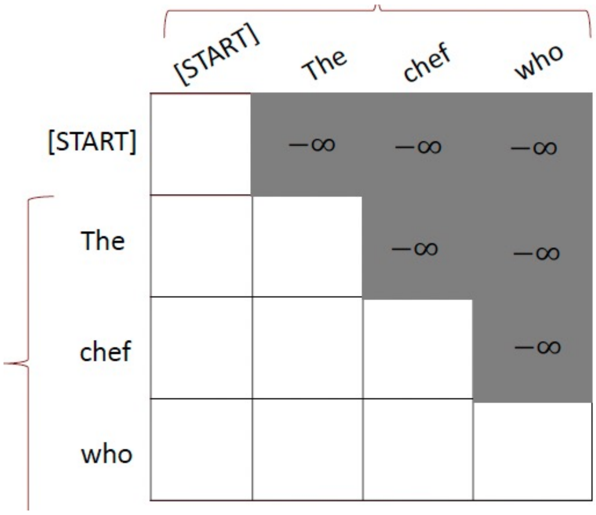

transformer block
transformer block
$\mathbf{v}_{\text{the}} + \mathbf{v}_1$
$\mathbf{v}_{\text{the}} + \mathbf{v}_4$
the
man
pets
the
cat
again

Figure 6.3.4. Multi-head attention mechanism: different heads learn to attend to different parts of the input

Position Encoding Because Transformers lack recurrence, they encode position with either fixed sinusoidal functions or learnable embeddings:

$$PE_{\text{pos},2i} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad PE_{\text{pos},2i+1} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$



	[START]	The	chef	who
[START]		$-\infty$	$-\infty$	$-\infty$
The			$-\infty$	$-\infty$
chef				$-\infty$
who				

Figure 6.3.5. Positional encoding

Auto-Regressive Masking In language generation, Transformers use **masked self-attention** to prevent future information leakage:

$$w_{ij} = \begin{cases} q_i^\top k_j, & j \leq i \\ -\infty, & j > i \end{cases}$$

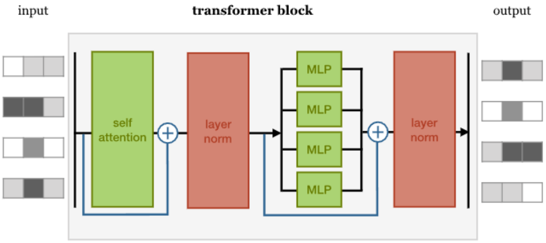


Figure 6.3.6. Masked self-attention for autoregressive decoding

Layer Normalization: It is a technique to speed up training and stabilize the network. Specifically, it works as follows,

- It normalizes hidden values to have zero mean and unit variance within each layer.
- Formulation:
  - Let $x \in \mathbb{R}^d$ be an individual(word) vector in the model.
  - Compute mean: $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$.
  - Compute std deviation: $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$.

  - Output:
$$\text{output} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\gamma + \beta$$
where $\gamma, \beta \in \mathbb{R}^d$ are learned parameters.

Transformer Components To sum up, each Transformer layer includes the following components,

- Multi-head attention
- Feed-forward networks
- Position encodings
- Layer normalization and residual connections

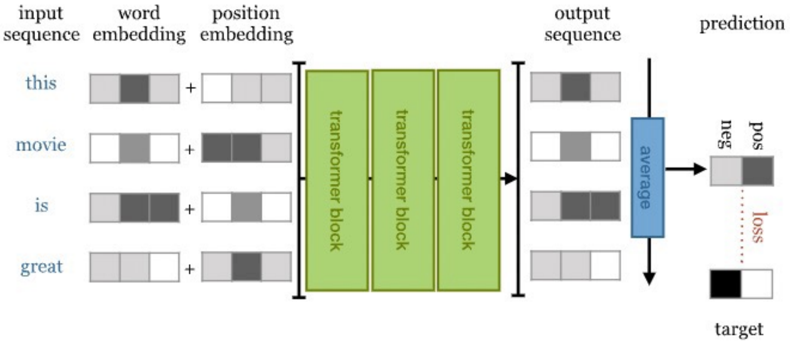
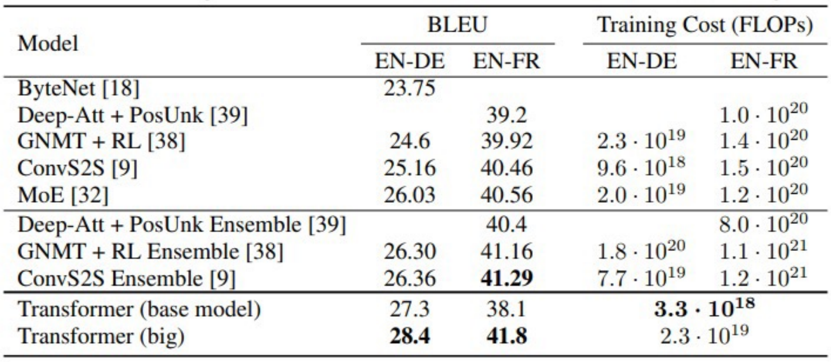


Figure 6.3.7. The Transformer encoder-decoder architecture

Classification Transformer Transformers can be directly applied to classification tasks. The typical workflow involves the following steps:

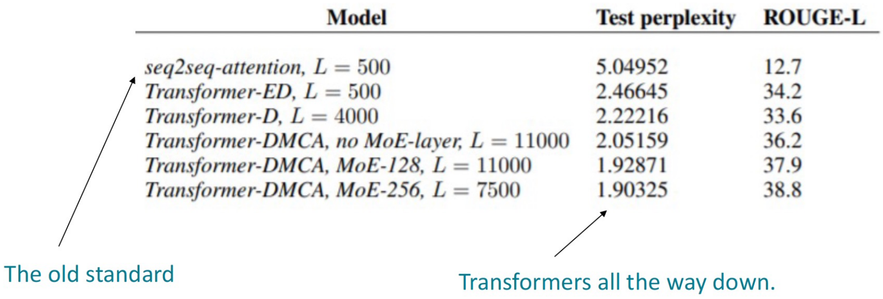
1. **Input Representation:** The input sequence is converted into word embeddings, which are then combined with positional embeddings to retain sequence order information.
2. **Transformer Processing:** The combined embeddings are passed through multiple Transformer blocks, which capture contextual relationships within the sequence.
3. **Output and Classification:** The processed sequence is fed into a classification head, often consisting of a linear layer followed by a softmax activation, to produce the final class predictions.

Performance of the Transformer It outperformed previous models such as GNMT and ConvS2S in both translation and document generation tasks (Vaswani et al., 2017).



Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Figure 6.3.8. BLEU scores and FLOPs: Transformer vs. other NMT models



Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard

Transformers all the way down.

Figure 6.3.9. Transformer performance on document generation tasks

6.4 Applications of Language Models in Economics and Social Science

The development of deep language models—ranging from LSTM-based architectures (e.g., ELMo/Stage II) to large-scale transformer models (e.g., BERT, RoBERTa, GPT III-IV)—has opened up powerful new methods for analyzing unstructured text at scale. These models capture complex patterns in language, enabling tasks such as classification, prediction, and interpretation with high accuracy and adaptability. In recent economic and social science research, these capabilities have been leveraged for a wide range of applications, illustrating how attention-based models contribute to empirical inquiry and policy analysis.

Detecting Disorganized Thought (FTD) in Mental Health. Sarzynska-Wawer et al. (2021) apply LSTM-based contextual embeddings (ELMo) to detect formal thought disorder (FTD) in patients. Their model outperforms traditional coherence-based approaches, which rely on dictionary-based NLP rules, by learning richer semantic representations from the language patterns of individuals. This suggests that language models can be used not just descriptively, but diagnostically, with implications for cognitive science and psychiatry.

Framing in Political Discourse. Card et al. (2022) investigate how language models can uncover ideological framing in political speech. Using a fine-tuned RoBERTa classifier, they label 140 years of U.S. congressional and presidential speeches according to their framing of immigration—pro-immigration, anti-immigration, or neutral. This enables a longitudinal study of how political rhetoric has evolved, demonstrating the role of transformer-based language models in historical and sociopolitical analysis.

Monitoring Remote Work in Labor Markets. Hansen et al. (2023) use DistilBERT to classify over one million job vacancy postings for their remote work (WFH) status. Their model identifies trends in remote work adoption by occupation, industry, and geography, showing dramatic growth in WFH postings since 2019. This highlights the usefulness of transformer-based classifiers for real-time monitoring of labor market dynamics using textual data.

Methodological Foundations: Text as Data. Gentzkow et al. (2019) provide a foundational review of text-as-data methods in economics. They discuss traditional bag-of-words models, topic modeling, and early machine learning approaches, laying the groundwork for newer embedding-based and attention-based models.

Current Challenges and Future Directions. Ash and Hansen (2023) propose a framework for how text algorithms are used in economics across four problem domains: measuring document similarity, detecting and relating concepts, and linking text with structured metadata. They identify two major challenges—validation and interpretability—and point to large language models as the next frontier in empirical research, particularly for tasks requiring semantic understanding and nuanced reasoning.

These applications illustrate the growing role of deep language models in transforming how economists and social scientists process and interpret language, policy, and discourse at scale.

6.5 The Notebooks

- The notebook Attention_Mechanism implements the attention mechanism and illustrates its connection to the classic non-parametric method(Nadaraya-Watson kernel regression);
- The notebook Transformer implements the transformer architecture.

7 Pretrained Transformers: BERT, GPT, and the Rise of Foundation Models

The development of pretrained transformer-based language models has redefined the landscape of modern natural language processing. Among these, BERT and GPT stand as foundational architectures that exemplify two contrasting yet complementary paradigms: **masked language modeling** (BERT) and **generative pretrained transformer** (GPT). Both models are built on the self-attention mechanisms introduced by the Transformer, but they differ in how they are trained, how they represent context, and how they are applied to downstream tasks:

- BERT (Bidirectional Encoder Representations from Transformers) uses a masked input to learn deep, bidirectional representations, making it particularly powerful for sentence-level understanding tasks such as question answering, natural language inference, and entity recognition;
- GPT (Generative Pretrained Transformer), by contrast, learns to predict the next token in a sequence in an autoregressive manner. This unidirectional setup enables fluent text generation and has proven essential for tasks like summarization, dialogue generation, and few-shot reasoning.

This section introduces both models in detail, contrasting their training objectives, architectural choices, and application domains. We also explore how their success has led to a broader class of *foundation models*—large-scale pretrained systems that can be adapted to a wide range of tasks with minimal supervision.

7.1 Pretraining Models

The success of modern NLP models like BERT and GPT relies fundamentally on a two-stage training strategy: **pre-training** followed by **fine-tuning**. Pre-training involves training a model on vast quantities of unlabelled text using self-supervised learning objectives. The aim is to compress knowledge from a massive, diverse corpus into the model's parameters, producing a general-purpose representation that can later be specialized to downstream tasks. At its core, pre-training is a process of scalable *compression* of linguistic and semantic information.

7.1.1 What is Pre-training?

In pre-training, the model learns to solve simple proxy tasks derived from the raw input itself, such as predicting missing or next words. This allows the model to extract syntactic and semantic features from language without requiring any human annotations.

- For encoder-based models like BERT, the objective is typically **masked language modeling (MLM)**: a percentage of input tokens are randomly masked, and the model must predict them using both left and right context.
- For decoder-only models like GPT, the objective is **causal language modeling (CLM)**: the model predicts the next token based only on previously seen tokens in a left-to-right fashion.

7.1.2 The Pre-training to Fine-tuning Pipeline

Pre-training produces a set of general-purpose parameters $\hat{\theta}$. Fine-tuning then adapts these parameters to specific NLP tasks using small, labeled datasets. Philosophically, pretraining can improve downstream NLP applications by serving as parameter initialization. The idea is to start from $\hat{\theta}$ and nudge the model toward solving a task-specific objective by minimizing an appropriate loss.

Mathematical view:

- **Pre-training:**

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\text{pretrain}}(x_i)$$

- **Fine-tuning:**

$$\theta^* = \hat{\theta} - \eta \nabla_{\theta} \mathcal{L}_{\text{task}}(x_j, y_j)$$

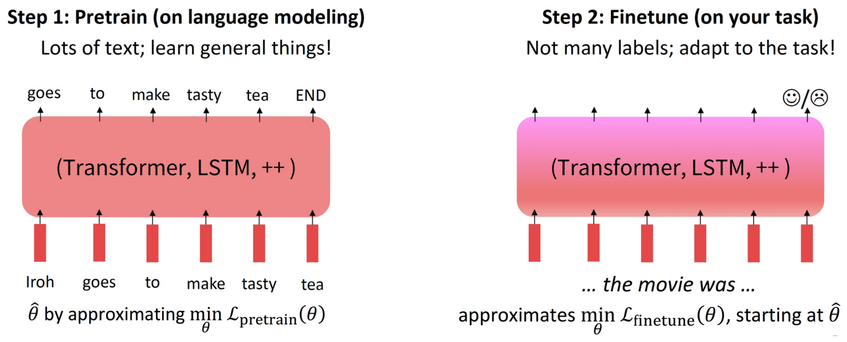
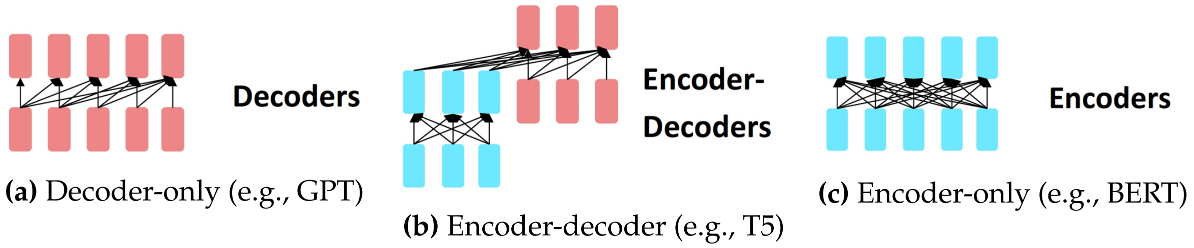

Step 1: Pretrain (on language modeling)
Lots of text; learn general things!
goes
to
make
tasty
tea
END
(Transformer, LSTM, ++)
Iroh
goes
to
make
tasty
tea
$\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{\text{pretrain}}(\theta)$
Step 2: Finetune (on your task)
Not many labels; adapt to the task!
☺/☹
(Transformer, LSTM, ++)
... the movie was ...
approximates $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$

Figure 7.1.1. Pre-training followed by task-specific fine-tuning

7.1.3 Architectural Variants in Pre-training

Transformer-based models adopt three broad architectural patterns during pre-training, each suited for different goals:



(a) Decoder-only (e.g., GPT)

(b) Encoder-decoder (e.g., T5)

(c) Encoder-only (e.g., BERT)

Figure 7.1.2. Three common pre-training architectures

(a) Decoder-only Models (e.g., GPT):

- Unidirectional: predicts next token based on previous tokens.
- Highly scalable; ideal for open-ended generation.
- Suitable for zero-shot and few-shot prompting tasks.

(b) Encoder-decoder Models (e.g., T5, BART):

- Encode input into a latent representation, then decode output.
- Supports both generation and understanding.
- Pre-trained via text-to-text transformations.

(c) Encoder-only Models (e.g., BERT):

- Bidirectional architecture: conditions on both left and right context.
- Optimized for understanding tasks: classification, NER, QA.
- Uses masked language modeling.

These architectures define not only how the model attends to input but also affect what kinds of tasks it excels at. In the next subsection, we explore how fine-tuning and prompting strategies adapt these pretrained models for specific applications. We start with BERT architecture.

7.2 BERT

7.2.1 The Architecture of Understanding

Since its introduction by Devlin et al. (2018), BERT (Bidirectional Encoder Representations from Transformers) has reshaped the way we model language. Its architecture is deceptively simple — a stack of Transformer encoders — but its impact lies in how it rethinks pretraining. Rather than predicting the next word in a sequence like traditional language models, BERT reads in both directions. It learns not by completing sentences, but by recovering corrupted fragments within them and reasoning over their coherence. This process is formalized through two interwoven training objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

Together, these tasks form a dual lens through which BERT perceives language: one lexical, the other logical.

7.2.2 Masked Language Modeling: Learning Through Obscurity

The cornerstone of BERT's representational power is its masked language model. The idea is simple, but profound: what if, instead of asking the model to generate the next word, we asked it to guess randomly obscured ones using the full context? Formally, let a sentence be tokenized into a sequence, where x_{CLS} is a special token always at the beginning and x_{SEP} is another special token used to separate two segments:

$$X = \{x_{[\text{CLS}]}, x_1, x_2, \ldots, x_n, x_{[\text{SEP}]}\}$$

A fixed proportion — 15% — of tokens is selected for masking. Among these selected tokens:

- 80% are replaced with the special token `[MASK]`;
- 10% are replaced with a random token from the vocabulary;
- 10% remain unchanged, though still treated as "masked" in loss computation.

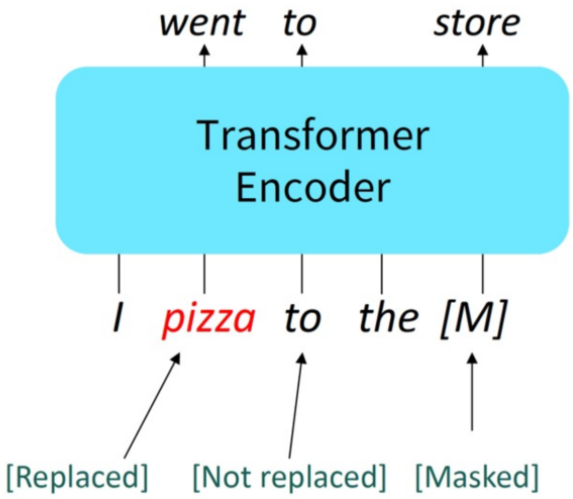


Figure 7.2.1. Masking

This stochasticity prevents the model from overfitting to a predictable masking pattern. It forces BERT to make robust inferences, even in cases where the input does not obviously signal corruption. To predict the masked tokens, BERT transforms each input into *contextual embeddings* through its stack of self-attention layers. For a masked token at position i, its output representation $y_i \in \mathbb{R}^d$ is projected to the vocabulary space via:

$$z_i = \text{softmax}(A_{\text{MLM}} y_i + b_{\text{MLM}}), \quad A_{\text{MLM}} \in \mathbb{R}^{V \times d}$$

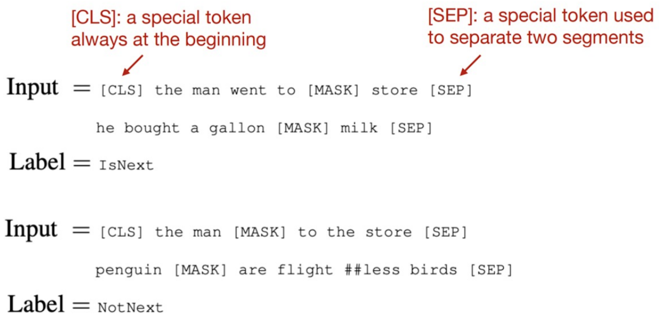
Here, z_i is a probability distribution over all words in the vocabulary V, and the *softmax* ensures these probabilities sum to one. The learning signal comes from the *cross-entropy loss*:

$$\mathcal{L}_{\text{MLM},i} = -\log(z_i[y_i^{\text{true}}])$$

where y_i^{true} is the correct word. The better the model's prediction, the closer the loss gets to zero.

7.2.3 Next Sentence Prediction: Modeling Coherence Across Sentences

While MLM helps BERT understand intra-sentence context, it alone cannot teach the model how sentences connect — how ideas flow in discourse. This is the motivation for *Next Sentence Prediction (NSP)*. In the NSP task, the model is fed two segments: Sentence A and Sentence B. It must predict whether B logically follows A in the original corpus. Half of the time, the sentence pairs are contiguous ("IsNext"); the other half, they are randomly sampled ("NotNext"). The sequence format is extended:



Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

Figure 7.2.2. Next Sentence Prediction

$$X = \{x_{[\text{CLS}]}, x_1, \ldots, x_n, x_{[\text{SEP}]}, x_{n+1}, \ldots, x_{n+m}, x_{[\text{SEP}]}\}$$

The [CLS] token is used to encode the relationship between the two segments. Its final hidden state y_{CLS} is passed through a *sigmoid classifier*:

$$z = \sigma(A_{\text{NSP}} y_{\text{CLS}} + b_{\text{NSP}}), \quad A_{\text{NSP}} \in \mathbb{R}^{1 \times d}$$

This output $z \in (0,1)$ is interpreted as the probability that Sentence B follows Sentence A. The loss, again, is *cross-entropy*:

$$\mathcal{L}_{\text{NSP}} = -D \log z - (1-D)\log(1-z)$$

where $D \in \{0,1\}$ is the binary ground truth.

7.2.4 *Joint Objective: Language at Two Scales*

What makes BERT powerful is not merely the existence of these two objectives, but their *joint optimization*. During pretraining, the model is taught to solve both tasks simultaneously:

$$\mathcal{L}_{\text{BERT}} = \sum_{i \in \mathcal{M}} \mathcal{L}_{\text{MLM},i} + \mathcal{L}_{\text{NSP}}$$

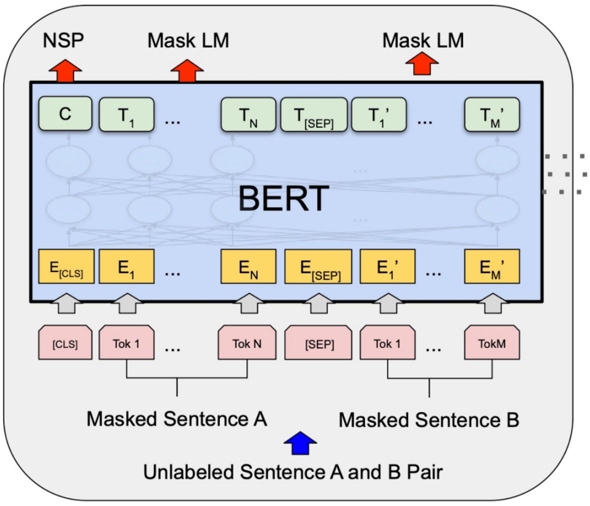


Figure 7.2.3. Pre-training Pipeline

Here, $\mathcal{M}$ is the set of masked token positions. This objective tunes BERT to understand language both at the granular level of words and the holistic level of discourse.

7.2.5 *Input Embeddings: Subwords, Segments, and Position*

embeddings are a foundational step that allows language models to bridge the gap between human-readable text and machine-readable numbers, while preserving the structure and meaning needed for downstream tasks. Specifically, each input token x_i is converted into a vector via three layers of embeddings:

- *Token Embeddings:* Represent each word or subword (e.g., ”transformation” → ”trans”, ”##form”, ”##ation”);

- *Segment Embeddings*: Distinguish between Sentence A and Sentence B;

- *Position Embeddings*: Preserve word order through learned position encodings.

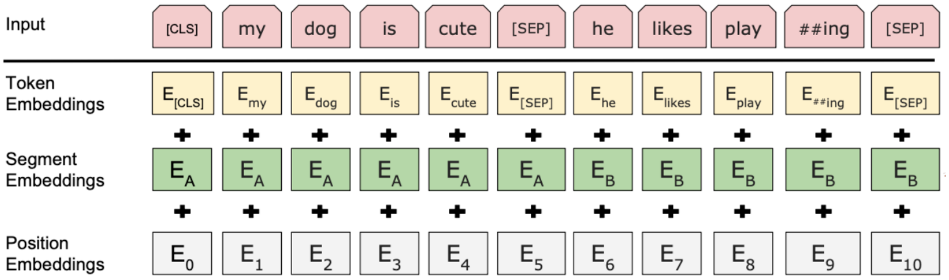
We will explain how the embedding works with the following diagram. This diagram illustrates the input embedding representation in the BERT (Bidirectional Encoder Representations from Transformers) model. Each token in the input sequence is represented by the sum of three types of embeddings: Token Embeddings, Segment Embeddings, and Position Embeddings.

Token Embeddings represent the semantic meaning of each word or subword unit in the input (e.g., [CLS], my, dog, is, cute, etc.). Notice that some words like playing are split into subword tokens such as play and ##ing.

Segment Embeddings are used to distinguish between different sentences in a pair. In the figure, tokens from the first sentence receive the embedding E_A, while tokens from the second sentence receive E_B.

Position Embeddings encode the position of each token in the sequence (e.g., E_0, E_1, ..., E_{10}), allowing the model to capture the order of words.

The final input representation for each token is obtained by summing its token, segment, and position embeddings. This combined embedding is then fed into BERT's Transformer layers, enabling the model to understand both the content and structure of the input.



Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

Figure 7.2.4. Input Embeddings

7.2.6 *Scaling and Training Regime*

BERT comes in two standard sizes:

- BERT-base: 12 layers, 768 hidden units, 12 attention heads, 110M parameters

- *BERT-large*: 24 layers, 1024 hidden units, 16 attention heads, 340M parameters

Its training is both data- and compute-intensive:

- Pretrained on Wikipedia (2.5B words) and BookCorpus (0.8B words)

- Sequence length: up to 512 tokens
- Batch size: scaled from 256 to 8,000 in later models like RoBERTa(Liu et al., 2019)
- Trained with 64 TPUs over 4 days

7.2.7 The Broader Picture: Understanding vs. Generating

BERT is an *encoder-only model* — its architecture is optimized for understanding, not generation. While it was initially superior to early GPT models in language understanding tasks, it lacks the autoregressive structure necessary for fluent text generation. However, what BERT gains in interpretability and depth of context, it trades off in output fluidity. This divide marks a philosophical split in modern NLP: do we want models that can understand, or ones that can speak?

At its core, BERT transforms the pretraining paradigm from linear prediction to *contextual reasoning*. It represents a shift from sequentially-driven models to those that read language more like humans do — with full awareness of what lies ahead and behind.

Its mathematical machinery — cross-entropy losses, softmax distributions, sigmoid classifiers — serves a larger purpose: to encode the structure of language itself. Each token prediction is not merely a classification problem; it is a window into how meaning is constructed, sentence by sentence, token by token, in the mind of a machine.

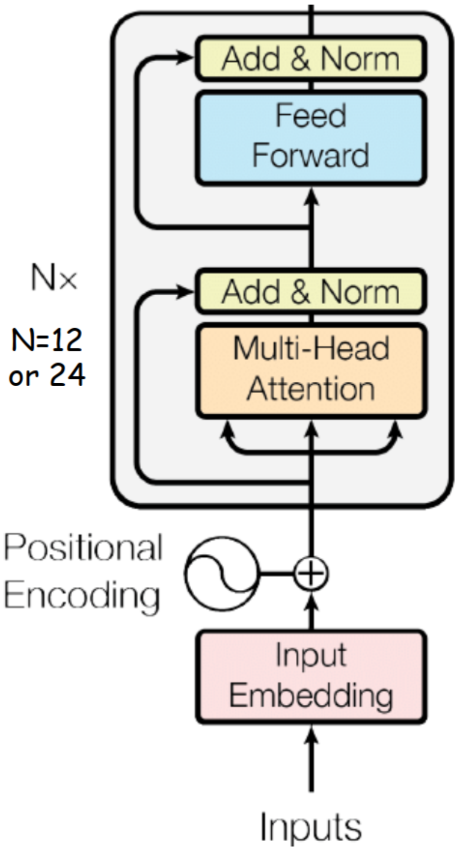


Figure 7.2.5. BERT Architecture - Encoder Only

7.2.8 Fine-tuning BERT

What follows the pretraining of the model is the fine-tuning. The idea of BERT fine-tuning is to make the pre-trained model work better on specific tasks by adjusting the pre-trained parameters.

BERT can be fine-tuned for two main types of tasks:

- Sentence-level tasks: where we assign a label to an entire sentence or a sentence pair (e.g., sentiment classification, entailment detection);
- Token-level tasks: where we assign a label to each token in the input (e.g., named entity recognition, part-of-speech tagging).

In sentence-level tasks, we typically use the output vector of the special [CLS] token to make predictions — this mirrors how BERT was pre-trained with the Next Sentence Prediction (NSP) objective. In token-level tasks, we use the output embeddings of individual tokens for prediction, similar to how BERT learns in Masked Language Modeling (MLM). After fine-tuning, BERT has achieved state-of-the-art performance on various NLP benchmarks, surpassing previous models and setting new standards on tasks such as:

- MNLI (Multi-Genre Natural Language Inference): predicting the relationship between two sentences (entailment, contradiction, or neutral);
- QQP (Quora Question Pairs): determining whether two questions are semantically equivalent;
- SST-2 (Stanford Sentiment Treebank): classifying the sentiment of a sentence as positive or negative;
- SQuAD (Stanford Question Answering Dataset): answering questions based on a given passage.

Transfer Learning in NLP and the Role of BERT The core idea of transfer learning is to leverage knowledge learned from solving one problem and apply it to another, often related, task. In natural language processing (NLP), transfer learning enables models to generalize across different datasets or objectives by transferring linguistic knowledge captured during pretraining.

Before BERT, transfer learning in NLP was primarily limited to the feature representation level. Models would use static word embeddings — such as those generated by word2vec or GloVe — as fixed input features for downstream tasks. These embeddings captured some semantic relationships between words, but they lacked context sensitivity and were not updated during task-specific training.

BERT fundamentally changed this paradigm. It introduced a pretraining-finetuning framework, where a deep bidirectional Transformer model is first pretrained on large-scale unlabeled corpora using unsupervised objectives (e.g., masked language modeling and next sentence prediction). Then, for each downstream task, BERT is fine-tuned end-to-end, adjusting all parameters — not just the final layers — using a smaller, labeled dataset.

This approach enables contextualized word representations and task-specific adaptation, significantly improving performance across a wide range of NLP benchmarks.

BERT in Multi-Task Learning A multi-task learning setup with BERT typically consists of two components:

- Shared layers: These are the pretrained BERT encoder layers (Transformer blocks), which capture general language representations across all tasks.
- Task-specific layers: These are lightweight, trainable layers added on top of BERT for each downstream task. They map the shared representation to task-specific outputs (e.g., classification logits or token tags).

The connection between the shared encoder and task-specific layers is often implemented via a linear transformation (i.e., a fully connected layer). During fine-tuning, both the shared encoder and task-specific layers are updated jointly or partially, depending on the setup.

7.2.9 Frontiers and Applications

FinBERT FinBERT (Liu et al., 2021) is a domain-specific adaptation of the BERT-base model. It was designed to address a critical limitation of the original BERT: its suboptimal performance on financial sentiment analysis tasks. While BERT was trained on general-purpose corpora such as Wikipedia and BookCorpus, financial language presents unique challenges — including specialized jargon, context-dependent polarity, and subtle expressions of uncertainty — that require tailored representation learning.

To address this, FinBERT was pretrained on a corpus of 4.9 billion tokens drawn from real-world financial texts:

- Corporate annual and quarterly filings from the SEC's EDGAR database (1994–2019),
- Financial analyst reports from the Thomson Reuters Intext database (2003–2012),
- Earnings conference call transcripts sourced from the Seeking Alpha platform (2004–2019).

The model was subsequently fine-tuned on a sentiment classification task involving 10,000 labeled sentences. These labels were distributed as follows: 36% positive, 46% neutral, and 18% negative. The goal was to predict the sentiment expressed in each sentence toward a financial entity or event.

FinBERT achieved superior performance compared to other models benchmarked in the same study — regardless of dataset size. Remarkably, when the training data was reduced to only 10% of its original size, FinBERT's accuracy only dropped from 88% to 80%, demonstrating both robustness and efficiency.

This performance illustrates a broader insight: *fine-tuning general-purpose pretrained models on domain-specific corpora can lead to significant gains in specialized tasks*. FinBERT serves as a compelling case study for this paradigm, and its success has inspired similar adaptations in fields ranging from biomedical text mining to legal document analysis.

Monetary Policy A particularly compelling application of BERT and its derivatives can be found in The Voice of Monetary Policy (Gorodnichenko et al., 2023). This study demonstrates how advances in NLP can be leveraged to quantify the tone and impact of central bank communication — a domain traditionally dominated by qualitative and narrative analysis.

The core idea of the paper is intuitive yet powerful: the *tone* of the Federal Reserve's monetary policy communications — particularly during Federal Open Market Committee (FOMC) press conferences — carries measurable informational value for financial markets. To quantify this tone, the authors first build a baseline model using a relatively simple neural network architecture: a multi-layer perceptron (MLP) with three hidden layers. This model is trained on manually labeled FOMC press conference transcripts, where each document is categorized based on its perceived tone (e.g., dovish vs. hawkish, positive vs. negative).

To evaluate the strength of more sophisticated NLP tools, the authors then compare this baseline model to transformer-based architectures — specifically:

- **BERT**, pretrained on general corpora (Wikipedia + BookCorpus),
- **RoBERTa**(Liu et al., 2019), a more robustly trained BERT variant with dynamic masking and more extensive pretraining data, and
- **FinBERT**(Liu et al., 2021), a domain-adapted BERT model fine-tuned on financial texts.

Each of these models is used to perform sentiment analysis on the same FOMC transcripts. The results are striking: not only do BERT-based models outperform the MLP in classification accuracy, but they also yield more stable and interpretable sentiment scores across time.

Moreover, the study links these sentiment scores to real-world financial outcomes. The authors find that a more optimistic tone — as captured by BERT and its variants — is consistently associated with positive stock market returns in the hours immediately following the press conference. This effect persists even after controlling for policy actions and macroeconomic fundamentals. In short, *how* the Fed communicates can be just as important as *what* it communicates.

The paper exemplifies several key takeaways for applied NLP in economics:

- Fine-tuned transformer models such as FinBERT can capture subtle domain-specific sentiment better than general-purpose models or shallow classifiers.
- Pretrained language models can be deployed directly in empirical economic analysis with minimal additional training.
- The textual tone — traditionally treated as a "soft" variable — can be quantitatively measured and linked to asset prices using modern AI tools.

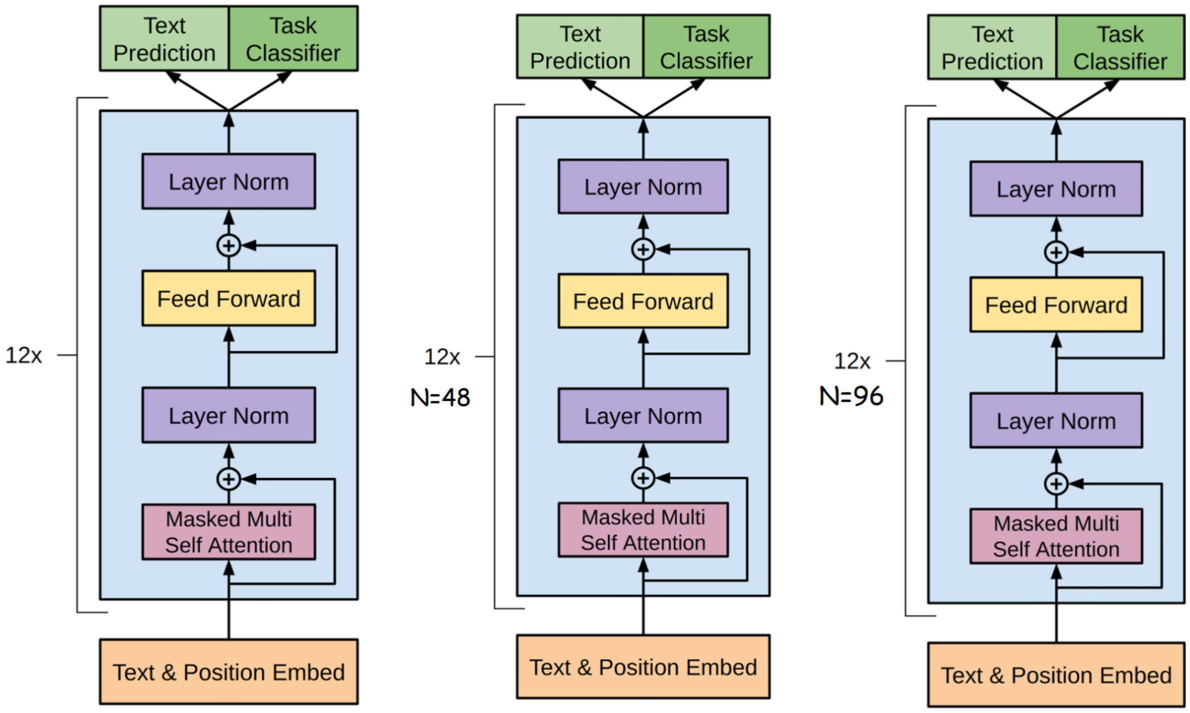
7.3 GPT

GPT, or *Generative* ***Pretrained*** *Transformer,* refers to a family of large-scale language models designed to perform a broad range of natural language understanding and generation tasks. In contrast to BERT, which is based on the encoder component of the Transformer architecture, GPT relies exclusively on the decoder stack.

The key characteristic of GPT is its autoregressive design: the model predicts the next word in a sequence based solely on the words that came before it. During training, it is exposed only to left-context tokens — the attention mechanism is masked such that no future tokens are visible. This setup enables GPT to naturally model text generation tasks, where the next word is sampled sequentially.

Importantly, GPT uses a single model for all tasks. Rather than using task-specific architectures, it leverages the same decoder stack — only scaled in depth, width, and data — making it an elegant and general-purpose approach to language modeling.

7.3.1 History of GPTs



(a) GPT-1 **(b)** GPT-2 **(c)** GPT-3

Figure 7.3.1. The History of GPTs

GPT-1 The first version, GPT-1 (Radford et al., 2018b), introduced a decoder-only Transformer with 12 layers, 768-dimensional hidden states, and a total of 117 million parameters. It was trained on the BooksCorpus, a dataset consisting of over 7,000 fiction books, providing long-range, coherent text examples.

GPT-1 was trained in a multi-task setting, where downstream tasks like classification, entailment, semantic similarity, and multiple choice were addressed by fine-tuning. The fine-tuning phase combined:

- Language modeling loss (predicting the next word);
- Supervised task loss (e.g., classification)

This dual-objective approach helped the model generalize better without overfitting to specific tasks.

GPT-2 GPT-2 significantly expanded the architecture, scaling to 48 layers, each with 1,600 hidden units and 25 attention heads, resulting in 1.5 billion parameters. Although architecturally similar to GPT-1, it represented an order-of-magnitude increase in both size and data diversity.

GPT-2 was trained on a high-quality, diverse corpus of 8 million web pages, enabling it to handle a wide array of topics and linguistic styles. One of GPT-2's key breakthroughs was its emergent capability: it could perform tasks like summarization, translation, and question answering with little or no fine-tuning — capabilities that were not explicitly trained for.

In empirical field, Reisenbichler et al. (2022) explores GPT-2's utility in Search Engine Optimization (SEO). In this context, the model generated webpage content optimized for search engines. Field experiments showed that GPT-2-generated content outperformed that created by human SEO experts — although the study's scale was limited, it underscored the practical potential of language models in marketing tasks.

GPT-3 GPT-3 (Brown et al., 2020a) represented a monumental leap in scale and capability. It increased the parameter count to 175 billion, using:

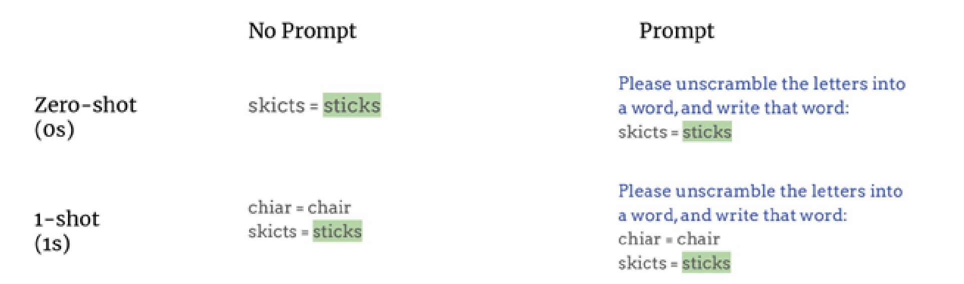
- 96 decoder layers;
- A context window of 2,048 tokens;
- Embedding dimensions of 12,288.

GPT-3 was trained on a massive 300-billion-token dataset that included Common Crawl, WebText2, Books, and Wikipedia. This unprecedented scale enabled it to excel at few-shot, one-shot, and even zero-shot learning. In these setups, the model could complete tasks simply by being shown a few examples (or just task instructions) in the input — without any gradient updates or architectural changes. This gave rise to a new paradigm in NLP known as **in-context learning**.

7.3.2 *In-Context Learning: Prompting Instead of Fine-Tuning*

Unlike earlier models that required retraining or parameter updates for each new task, GPT-3 popularized a user-friendly approach: treating the model as a fixed black box and interacting with it through carefully designed input prompts.

In-context learning allows users to simply "show" the model a few examples directly within the input. For instance:



	No Prompt	Prompt
Zero-shot (0s)	skicts = sticks	Please unscramble the letters into a word, and write that word: skicts = sticks

1-shot
(1s)
	chiar = chair
	skicts = sticks
		Please unscramble the letters into a word, and write that word:
		chiar = chair
		skicts = sticks

Figure 7.3.2. In-context Learning

This approach requires no additional training, making powerful language modeling accessible without large-scale infrastructure or labeled data.

7.3.3 *Pretraining Data for LLMs: The Hidden Engine Behind GPT*

While architectural innovations and model scaling have fueled the success of GPT, another critical — yet often less visible — ingredient is the *quality and scale of pretraining data*. In autoregressive models like GPT, the ability to generate coherent, factually grounded, and contextually appropriate text stems from exposure to an enormous and diverse corpus of unlabeled language data.

Modern GPT-style models are trained on **unlabeled** data at a truly massive scale: often in the order of trillions of tokens. In comparison, labeled datasets — used for supervised fine-tuning — are several orders of magnitude smaller and more expensive to obtain (e.g., 50 million labeled vs. 240 trillion unlabeled tokens). This gap underscores the value of effective pretraining: models learn to reason, summarize, translate, and more, largely from the statistical patterns found in raw text alone.

A recent paper by Soldaini et al. (2024) sheds light on this pipeline by introducing *Dolma*, a transparent, trillion-token dataset curated by Hugging Face. It serves as a key example of how modern large-scale datasets are constructed for LLM training, and how they could be leveraged by future open-source GPT-style models.

Where Does the Data Come From? The pretraining corpus for GPT models typically combines several broad categories:

- **Websites** (e.g., Common Crawl)

- **Books** (fiction and non-fiction)
- **Scientific articles**
- **Social media and discussion forums**
- **Code repositories**

The process usually begins with URL harvesting. However, not all internet content is appropriate for training. Hence, the next steps involve:

- **Toxicity filtering**: Identifying and removing harmful, biased, or offensive content.
- **Language filtering**: Ensuring consistency in the language distribution across documents.
- **Deduplication**: Preventing repetition, which could bias the model.
- **PII redaction**: Scrubbing personally identifiable information to preserve user privacy.

While organizations like Hugging Face document and share their data pipelines openly, not all models are as transparent. Proprietary models like **GPT-3** or **DeepSeek** often do not release their full training corpus. As a result, there is frequent speculation about undocumented data sources — such as whether DeepSeek uses forums like Baidu Tieba to enhance its fluency in Chinese. The lack of reproducibility creates challenges in attribution and fairness evaluation.

Data Ablation: Understanding What Matters An important technique used in GPT pretraining research is *data ablation*. The idea is to train smaller models on subsets of curated data — varying in domain, source, or preprocessing — and evaluate their performance on standard benchmarks, especially:

- Zero-shot and few-shot **in-context prompting**
- Question answering
- Knowledge recall tasks

By systematically ablating or removing certain data components, researchers can identify which sources are most valuable. For instance, is Common Crawl still helpful after heavy filtering? Do books add long-range coherence? Does Reddit-style dialogue improve conversational fluency?

Once effective data sources and strategies are identified, they can be scaled to build larger and more capable GPT-style models. This method supports the growing trend of **data-centric AI** — optimizing what the model learns from, not just how it learns.

Toward Open and Reproducible GPT Models The pretraining data pipeline plays a foundational role in the behavior, safety, and capability of LLMs. As GPT-like models continue to proliferate, the call for transparent data practices is growing. Initiatives like Hugging Face's *Dolma*, OpenAI's partial disclosures, and EleutherAI's *The Pile* point toward a future where high-quality, diverse, and responsibly curated corpora become as important as the models themselves.

In this light, the success of GPT is not merely a triumph of architecture or compute — it is also a testament to the power of scale, diversity, and design in language data.

7.3.4 Tokenization

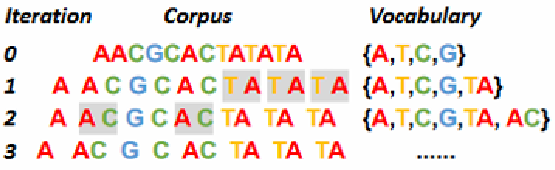
Tokenization is the process of splitting raw text into smaller units called *tokens*, which serve as the atomic elements processed by language models. On average, one token corresponds to approximately 0.75 words in English. For example, the sentence `"I love AI."` would typically be tokenized into four tokens: `["I", "love", "A", "I"]` or similar, depending on the tokenizer used.

Most modern large language models — including GPT, BERT, and LLaMA — adopt a technique known as **Byte-Pair Encoding (BPE)** or its variants. BPE is a data-driven, subword tokenization algorithm that strikes a balance between character-level precision and word-level efficiency.

How BPE Works: The algorithm begins with a base vocabulary containing individual characters or common symbols. Then it proceeds through an iterative process:

1. Count all symbol pairs in the corpus (e.g., letter pairs or byte pairs).
2. Identify the most frequent pair (e.g., `TA`) and merge it into a new token.
3. Add this merged pair to the vocabulary.
4. Repeat the process until a predefined vocabulary size is reached.

To illustrate, consider the following DNA sequence. Initially, the vocabulary consists of four symbols: {`A, T, C, G`}. If the sequence `TA` appears frequently, it is added to the vocabulary as a new token. Later, other frequent subsequences such as `AC` or `CG` may also be merged.



Iteration	*Corpus*	*Vocabulary*
0	AACGCACTATATA	{A,T,C,G}
1	A A C G C A C T A T A T A	{A,T,C,G,TA}
2	A A C G C A C TA TA TA	{A,T,C,G,TA, AC}
3	A AC G C AC TA TA TA	......

Figure 7.3.3. Illustration of BPE vocabulary construction using a DNA sequence

This method is highly flexible. It captures full words (e.g., `machine`) and common subwords (e.g., `##ization`) while breaking rare or novel words into known fragments (e.g., `trans||former||ify`). Such granularity allows models to handle out-of-vocabulary words and neologisms gracefully.

Trade-offs Between Efficiency and Effectiveness: Tokenization design involves a trade-off between:

- **Efficiency:** Smaller vocabulary sizes lead to smaller embedding matrices and less computational overhead.
- **Effectiveness:** Larger vocabularies can encode more semantically meaningful units, reducing the number of tokens per sentence.

Character-level tokenization provides maximum flexibility but is often inefficient and slow to train. Conversely, large vocabularies reduce sequence length but increase memory usage and training cost. Therefore, most models strike a balance by only merging subwords or phrases that occur frequently and have distinct meanings.

Vocabulary Sizes in Practice: Vocabulary size varies depending on the model and its use case:

- For general-purpose monolingual models such as **GPT** and **LLaMA**, the vocabulary size typically ranges between **30,000 to 50,000** tokens.
- In production-scale multilingual or domain-specific systems, the vocabulary may expand to **over 250,000** tokens to ensure coverage across diverse languages or technical content.

Choosing an appropriate tokenization strategy is essential for building scalable and effective language models. As computational resources grow, future models may adopt even more adaptive tokenization schemes that better align with human language structure and efficiency requirements.

7.3.5 Compute-Efficient Training with GPUs

Training large-scale language models such as GPT-3 and DeepSeek-V3 depends not only on data and architecture but also on computational efficiency. The core of this efficiency lies in the use of GPUs — massively parallel processors that significantly accelerate tensor-based workloads (Wikipedia contributors, 2023). Unlike CPUs, which feature a handful of powerful cores, GPUs offer thousands of lightweight cores optimized for simultaneous execution, making them ideal for operations like matrix multiplications that dominate Transformer-based models.

Memory Hierarchy. GPU architecture is supported by a hierarchical memory system that governs data flow and computational throughput:

- **SRAM (L1/L2 cache)**: Limited in capacity but extremely fast (typically 20MB, 19 TB/s), which is used for on-chip reuse.
- **High Bandwidth Memory (HBM)**: Main GPU memory (typically 40GB, 1.5 TB/s). It's typically Used for storing weights, activations, gradients.
- **CPU DRAM**: Very large but slower(typically more than 1TB, 12.8GB/s). Used for swapping when GPU memory overflows.

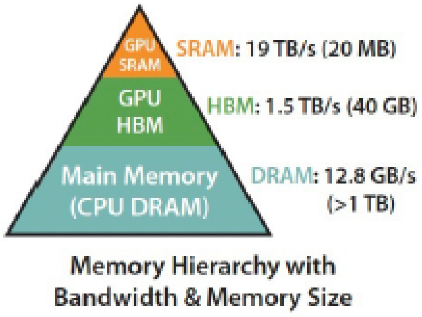


Figure 7.3.4. Hierarchy of GPU and CPU memory

Numerical Precision. To reduce computational load and memory usage, modern LLM training pipelines utilize lower-precision formats. This transition enables higher throughput at the cost of some precision:

- **FP32**: Standard 32-bit float. Accurate but compute-heavy.
- **BF16 / FP16**: Mixed-precision formats with reduced bit-width, commonly used in training (Micikevicius et al., 2018).
- **TF32**: Used on NVIDIA A100; approximates FP32 range with FP16 mantissa.
- **FP8**: Ultra-low precision; DeepSeek-V3 is among the first models to use it for training (DeepSeek, 2024).

Lower-precision formats reduce memory footprint, improve training speed, and often act as regularization by injecting mild numerical noise. Their use has become essential for scaling models beyond hundreds of billions of parameters.

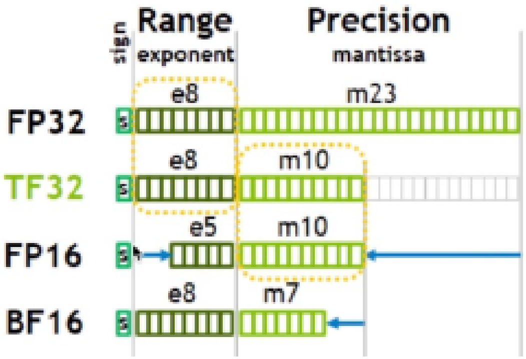


Figure 7.3.5. Comparison of numerical formats

Parallelism and Batch Strategy. PyTorch enables GPU-accelerated computation and supports distributed training through modules like Distributed Data Parallel (DDP), allowing models to scale across multiple GPUs and machines. To maximize GPU utilization:

- Use large batch sizes when memory allows.
- Apply gradient accumulation to simulate larger batches under memory constraints.
- Scale learning rate proportionally with batch size to maintain stable convergence.

Optimizers and Schedulers. The *AdamW* optimizer is widely used for GPT-style training, as it decouples weight decay from the adaptive update rule (Loshchilov and Hutter, 2019). Learning rate warm-up and cosine decay schedules are common practices to ensure smooth optimization. Additionally, correct initialization is essential; for instance, *He initialization* works well with ReLU activations and helps maintain stable gradients during early training (He et al., 2015).

Attention Efficiency. One of the most computationally expensive components in Transformer models is the attention mechanism, which traditionally scales quadratically in both time and memory. **FlashAttention** (Dao et al., 2022) addresses this by reformulating attention as a fused, IO-aware kernel. It reduces redundant memory transfers and avoids allocating large intermediate buffers. FlashAttention is now available in PyTorch and is particularly beneficial for models trained with long context windows.

7.3.6 Mixture of Experts (MoE)

Mixture of Experts (MoE) is a sparse neural network architecture where only a subset of parameters are activated for each input. Unlike dense models that apply the full model to every token, MoE selectively routes tokens to different expert subnetworks, making it possible to scale the total number of parameters without increasing compute per token (Wikipedia, 2023). This selective activation enables compute-efficient scaling. For instance, DeepSeek-V3 has 671 billion total parameters, but each input token activates only around 37 billion, significantly reducing memory and FLOPs during training (DeepSeek, 2024).

Architectural Contrast between Dense vs. Sparse Models Dense models utilize all weights regardless of the input. In contrast, MoE models activate only a small number of experts, which are specialized sub-networks. This reduces redundant computation and allows targeted processing.

Aspect	Dense Model	Sparse Model (MoE)
Parameter Usage	All parameters used	Only a subset of experts activated
Efficiency	High computation cost	Lower compute per forward pass
Scalability	Memory-bound	Efficient scaling to trillions of parameters

Table 9. Dense vs. Sparse Models: Key Differences

MoE Components. An MoE model consists of:

- A set of expert networks, typically MLPs.
- A gating network that routes input tokens to the most relevant experts.
- A mechanism to combine the outputs of the selected experts.

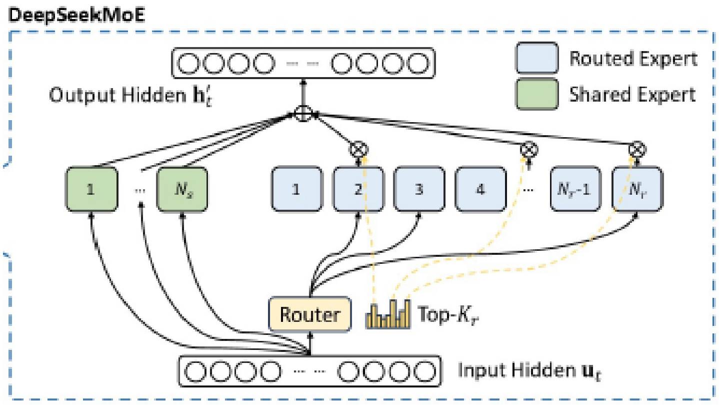


Figure 7.3.6. MoE Layer Architecture (adapted from (DeepSeek, 2024))

Shared vs Routed Experts. Some models, such as DeepSeek-V3, use shared experts that are always active for every token, in addition to sparsely routed experts. This hybrid design improves generalization while maintaining specialization.

Mathematical Formulation. Each expert E_i performs a transformation on the input:

$$E_i(x) = W_i x + b_i$$

The gating network computes a relevance distribution:

$$G(x) = \text{Softmax}(W_g x + b_g)$$

The top-K experts are selected based on the highest gating scores, and the final output is:

$$y = \sum_{i \in \text{Top-K}} G_i(x) E_i(x)$$

Load Balancing. To avoid expert overuse or underuse, MoE introduces a load-balancing loss:

$$L_{\text{balance}} = H(G(x)) + \lambda \sum_i \left(\sum_x G_i(x) - \frac{1}{N} \right)^2$$

where $H(G(x))$ is the entropy of the gating distribution, and the second term encourages uniform expert utilization.

Benefits of MoE

- **Efficient Scaling.** MoE architectures allow models to scale to trillions of parameters while keeping the active computation per token constant. This enables high-capacity models to run on existing infrastructure.
- **Compute-Aware Design.** By activating only a small fraction of the full model, MoE makes more efficient use of memory, bandwidth, and compute resources. This is crucial for cost-effective deployment at scale.
- **Adoption in SOTA.** MoE has been adopted by many cutting-edge models including GPT-4, DeepSeek-V3, Qwen, Grok, Gemma, and Mistral (DeepSeek, 2024; Wikipedia, 2023). These models demonstrate that MoE not only enhances efficiency but can improve downstream performance through expert specialization.

7.3.7 Native Sparse Attention (NSA)

Motivation. Self-attention mechanisms scale quadratically with sequence length, making them inefficient for long-context language models. While traditional sparse attention reduces this cost, it often introduces irregular memory access patterns that are not friendly to GPU hardware.

NSA vs Traditional Sparse Attention. Native Sparse Attention (NSA) (Li et al., 2024) addresses these limitations by introducing a hardware-aware sparse attention mechanism. Unlike older sparse attention approaches that focus solely on reducing FLOPs, NSA integrates sparsity into the kernel design, optimizing both memory access and parallelism.

Feature	**Traditional Sparse Attention**	**NSA**
Usage	Typically used only at inference	Trainable and inference-ready
Kernel Design	Standard attention blocks	Redesigned with **Triton** for hardware efficiency
Efficiency	Memory savings but not optimized for training	**Fully optimized for both training & inference**
Performance	Helps long-context models but slower training	Improves training and inference speeds

Table 10. Comparison between Traditional Sparse Attention and NSA (Li et al., 2024)

Key Design: Dynamic Token Selection. Instead of computing full attention over the entire sequence, NSA dynamically selects important key-value tokens per query. This compression step allows the model to focus only on relevant context while reducing compute.

Hardware-Efficient Kernel. NSA introduces a grid-aligned CUDA kernel that organizes memory accesses in a warp- and cache-friendly manner. By aligning the sparse attention layout with GPU execution patterns, NSA significantly improves memory throughput and utilization.

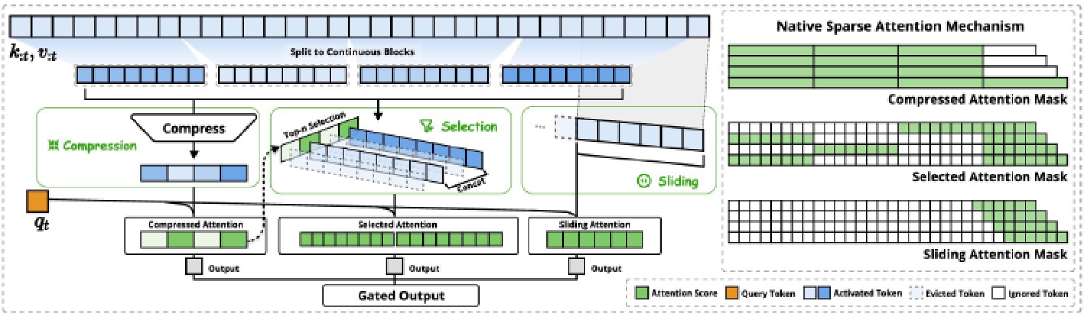


Figure 7.3.7. NSA Architecture: Token Selection + Grid-based Computation

Benefits. NSA offers three primary advantages:

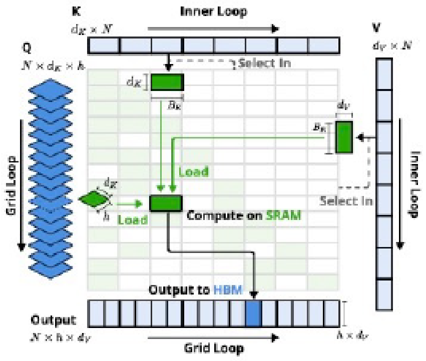


Figure 7.3.8. Grid-based Kernel Design for Memory Efficiency

- **Faster Training**: By avoiding dense attention computations, NSA accelerates training and inference, especially for long sequences.
- **Better Long-Context Handling**: The model retains accuracy even with sparse context windows, making it ideal for summarization and long-form QA.
- **Efficient Memory Usage**: Reduces memory footprint by minimizing unnecessary key-query interactions and intermediate activations.

7.4 The Notebooks

- The notebook BERT_Hugging_Face leverages the HuggingFace API for BERT to conduct a simple sentiment classification task;
- The notebook BERT_Finetuning fines a BERT Model on economic text, adapted from the github repository.

8 Posttraining LLMs

Posttraining refers to the critical transformation that a large language model (LLM) undergoes after its initial pretraining phase. While pretraining equips the model with general knowledge and linguistic fluency by exposing it to vast amounts of internet text, it often results in behavior that is misaligned with human values or goals. This phenomenon is known as **AI misalignment**. Models at this stage may hallucinate facts, exhibit unsafe biases, or fail to follow user instructions reliably.

8.1 Motivation and Scope

8.1.1 Why Posttraining Matters

The goal of posttraining is to bridge the alignment gap between a model's general capabilities and the specific behaviors we want in real-world applications. Unlike pretraining, which is unsupervised and computationally expensive, posttraining is typically more targeted and resource-efficient. However, its effectiveness depends on the foundation laid during pretraining.

Posttraining teaches models how to:

- Follow human instructions.
- Refuse unsafe or inappropriate requests.
- Use external tools like search engines, APIs, or calculators.
- Emulate specific personas or tones.

A well-executed posttraining process can turn a powerful base model into a reliable assistant, coder, tutor, or researcher. Yet, excessive alignment can also lead to ″over-correction,″ making the model overly cautious or evasive, thus reducing its utility.

8.2 Core Techniques in Posttraining

8.2.1 Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning (SFT) is the critical first phase of posttraining that directly teaches an LLM how to behave in ways that are aligned with human preferences. While pretraining endows the model with broad linguistic and factual knowledge, it does not teach the model how to follow instructions, maintain tone, or behave safely and consistently. SFT fills this gap through explicit behavioral cloning — training the model on high-quality (input, output) instruction-response pairs.

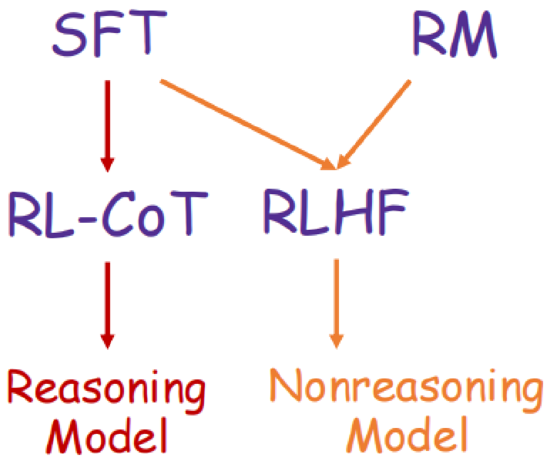


Figure 8.1.1. Core components of LLM posttraining.

Data Construction. The effectiveness of SFT heavily depends on the quality and diversity of the data. These datasets typically include tens of thousands to hundreds of thousands of examples, covering a wide range of user intents and task types. Tasks commonly include:

- Closed-form question answering (e.g., "What is the capital of France?")
- Open-ended instruction following (e.g., "Write a short poem about hope.")
- Text classification or extraction
- Summarization, translation, and reasoning

While some examples are manually curated by expert annotators, many instruction-response pairs are generated using existing LLMs (e.g., GPT-4), and then cleaned, filtered, or revised by humans. This process enables rapid dataset construction while maintaining quality.

Training Objective. SFT typically uses a standard supervised cross-entropy loss. The model is trained to predict the next token in the reference response given the full input prompt. The goal is not to "explore" or optimize for rewards, but to faithfully mimic high-quality demonstrations. Because of this, SFT can be understood as a form of behavioral cloning from ideal behavior.

Initialization for Alignment. Beyond its standalone utility, SFT serves as the starting point for more advanced alignment methods such as RLHF and DPO. Without SFT, reinforcement learning would lack a stable and reasonable initialization and would often fail to converge or produce chaotic behavior.

Evaluation. SFT models are evaluated using both automatic and human-centric metrics. One of the most commonly used benchmarks is MMLU (Massive Multitask Language Understanding) (Hendrycks et al., 2021), which measures zero-shot accuracy across 57 tasks spanning law, mathematics, medicine, history, and more. High MMLU scores indicate that the model has preserved its factual and reasoning capabilities after alignment.

Challenges. Despite its strengths, SFT has notable limitations:

- It cannot capture open-ended preferences like creativity, humor, or nuance.
- It assumes the labeled output is the only acceptable response, discouraging diversity.
- It lacks the ability to distinguish between outputs that are correct but phrased differently.

Thus, while essential, SFT is typically followed by reinforcement-based techniques (e.g., RLHF) to further refine and personalize model behavior.

In Practice. Models like InstructGPT and early versions of ChatGPT underwent extensive SFT using proprietary instruction datasets. Open-source initiatives like OpenAssistant and Alpaca have also demonstrated that with as few as 50K examples, meaningful alignment can be achieved in smaller models. Nevertheless, performance plateaus quickly with SFT alone, which is why it is best viewed as the first step — not the final word — in posttraining.

8.2.2 Parameter-Efficient Fine-Tuning (PEFT)

Large language models (LLMs) often contain billions of parameters, making full fine-tuning computationally expensive and memory-intensive. In many real-world use cases—especially in academia, enterprise, or edge deployment—fine-tuning all parameters is neither necessary nor feasible. Parameter-Efficient Fine-Tuning (PEFT) addresses this challenge by updating only a small, targeted subset of model parameters, enabling fast, low-resource adaptation with minimal performance degradation.

Motivation. LLMs are highly overparameterized, and much of their knowledge is stored in shared representations. When adapting an LLM to a new task, domain, or tone, it is often sufficient to adjust only a small part of the network to redirect these general capabilities toward specific goals. PEFT leverages this insight to make fine-tuning more accessible, especially for smaller institutions and open-source developers.

LoRA (Low-Rank Adaptation). LoRA, introduced by Hu et al. (2022), is one of the most popular and widely adopted PEFT techniques. It freezes the original model weights and injects small trainable matrices into the attention or MLP layers. Instead of updating the full weight matrix $W \in \mathbb{R}^{d \times d}$, LoRA learns a low-rank approximation of the incremental weight matrix ΔW via two smaller matrices B and A:

$$\Delta W = \alpha B A$$

,where α is the tradeoff between the pretrained knowledge and the task-specific knowledge, $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, $r \ll d$. These matrices are added to the original projection during training, allowing efficient adaptation without touching the backbone model. LoRA layers can be added to a subset of Transformer blocks (e.g., only to attention projections), which further reduces training and memory overhead.

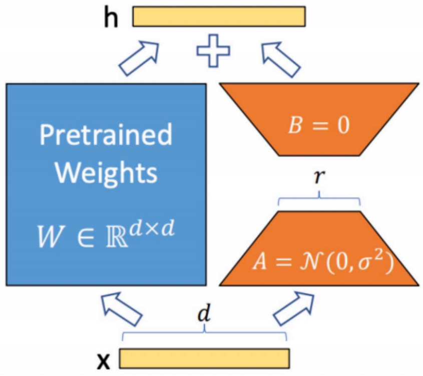

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 8.2.1. LoRA: Low-Rank Adaptation in Transformer layers.

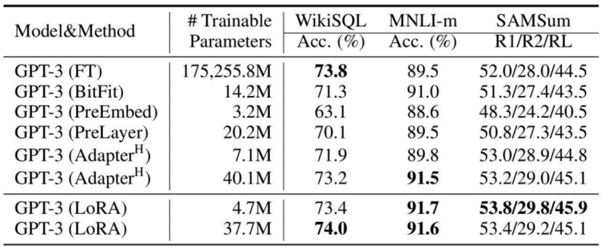
QLoRA. QLoRA (Dettmers et al., 2023) builds upon LoRA by quantizing the base model to 4-bit precision, drastically reducing GPU memory usage while preserving accuracy. It introduces a custom 4-bit NormalFloat (NF4) quantization scheme and uses a double quantization technique to compress the weight matrices further.

QLoRA also applies quantization-aware memory management and gradient checkpointing to enable fine-tuning of very large models (e.g., 65B parameters) on a single consumer-grade GPU. Despite the aggressive compression, QLoRA achieves performance comparable to full-precision LoRA and full fine-tuning.

Performance and Practical Benefits. Empirical studies show that LoRA and QLoRA can match or even outperform full fine-tuning in many instruction-following or task-specific settings, especially when compute is constrained. Their advantages include:

- **Lower memory usage:** Enabling fine-tuning of large models on consumer hardware.

- **Faster training and inference:** Due to fewer updated parameters.
- **Modularity:** LoRA weights can be stored and deployed separately from the base model, enabling dynamic task switching.
- **Privacy and localization:** Organizations can fine-tune LLMs on private datasets without touching sensitive base weights.



Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^H)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^H)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Figure 8.2.2. LoRA achieves performance comparable to full fine-tuning.

Use in the Open-Source Community. LoRA and QLoRA have become essential tools in the open-source ecosystem. Models like Alpaca, Vicuna, and Mistral-Instruct were built using LoRA layers on top of base checkpoints. Their simplicity, efficiency, and strong empirical performance make them ideal for domain adaptation, task specialization, and local deployment in privacy-sensitive environments.

8.2.3 Reinforcement Learning from Human Feedback (RLHF)

While Supervised Fine-Tuning (SFT) provides the foundational behavior of instruction following, it is inherently limited in its ability to represent complex, nuanced, or context-sensitive human preferences. SFT teaches a model to mimic what it sees, but it does not prioritize preferred completions over suboptimal ones. This is where Reinforcement Learning from Human Feedback (RLHF) plays a crucial role.

RLHF augments SFT by introducing dynamic learning signals from human judgments. Rather than training on a fixed dataset of correct answers, the model learns to optimize for outputs that humans actually prefer, even in cases where multiple reasonable completions exist.

The RLHF Pipeline. The standard RLHF process, popularized by InstructGPT and ChatGPT, involves three sequential stages:

1. **Supervised Fine-Tuning (SFT):** The model is first fine-tuned to follow instructions using a curated instruction-response dataset. This establishes a reasonable initialization that avoids dangerous or incoherent completions during exploration.

2. **Reward Modeling (RM):** Human annotators are presented with multiple completions from the SFT model for the same prompt. They rank or rate these completions, and this comparison data is used to train a separate reward model. This reward model learns to approximate human preference by assigning scalar values to outputs.

3. **Reinforcement Learning (PPO):** Using the reward model as a proxy for human judgment, the model is further fine-tuned via Proximal Policy Optimization (PPO) (Schulman et al., 2017). The goal is to maximize the expected reward while preventing the model from drifting too far from the original SFT behavior. This is done by constraining the update through a KL penalty $D_{KL}(\pi_\theta(y|x)||\pi_{SFT}(y|x)$, which measures the distance between the learned policy $\pi_\theta(y|x)$ and the SFT policy $\pi_{SFT}(y|x)$:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)} \left[r(x,y) - \beta D_{KL} \left(\pi_\theta(y|x) \| \pi_{\text{SFT}}(y|x) \right) \right]$$

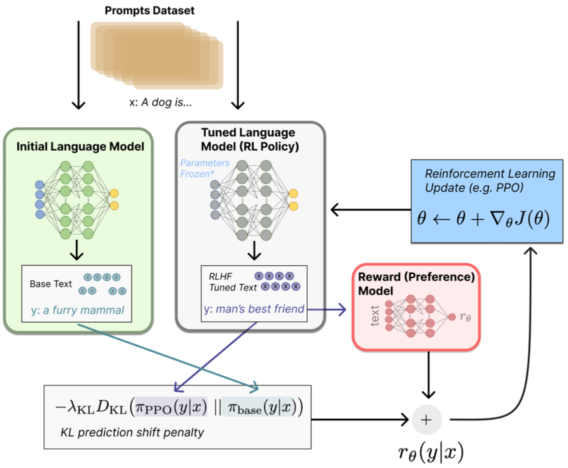

Prompts Dataset
x: A dog is...
Initial Language Model
Base Text
y: a furry mammal
Tuned Language Model (RL Policy)
Parameters Frozen*
RLHF Tuned Text
y: man's best friend
Reinforcement Learning Update (e.g. PPO)
$\theta \leftarrow \theta + \nabla_\theta J(\theta)$
Reward (Preference) Model
text
r_θ
$-\lambda_{\text{KL}} D_{\text{KL}} (\pi_{\text{PPO}}(y|x) \; || \; \pi_{\text{base}}(y|x))$
KL prediction shift penalty
+
$r_\theta(y|x)$

Figure 8.2.3. PPO optimizes SFT policy with RM guidance.

Advantages. RLHF improves response helpfulness, reduces unsafe or toxic outputs, and adapts to subtler human preferences (e.g., politeness, creativity, clarity). It enables LLMs to move beyond rigid instruction-following to more context-aware, human-aligned behavior. Notably, InstructGPT demonstrated that a smaller model trained with RLHF could outperform a much larger model trained only with SFT in human preference evaluations.

Limitations and Challenges. Despite its impact, RLHF is complex and resource-intensive. Several notable limitations include:

- **Reward hacking:** The model may learn to exploit loopholes in the reward model without truly improving quality (Weng, 2024).
- **Instability:** PPO fine-tuning can be unstable, especially if the reward model is noisy or biased.
- **High data cost:** Reward modeling requires expensive human comparisons and careful quality control.
- **Misalignment of the proxy:** The reward model may fail to capture nuanced values like fairness, originality, or truthfulness.

Real-World Deployment. RLHF has become a cornerstone of modern AI assistant development. OpenAI's ChatGPT, Anthropic's Claude, and DeepMind's Sparrow all employ RLHF-like pipelines. While alternatives like Direct Preference Optimization (DPO) are gaining attention (see next subsection), RLHF remains the most widely adopted technique for aligning LLMs with human values in production.

8.2.4 Direct Preference Optimization (DPO)

Direct Preference Optimization (DPO) (Rafailov et al., 2023) has emerged as a compelling alternative to traditional RLHF pipelines. It simplifies the posttraining process by removing the need to train a separate reward model and bypassing reinforcement learning altogether. Instead, DPO directly optimizes the LLM's policy using preference data collected from human comparisons.

Core Insight. Rather than transforming human preferences into a scalar reward signal via a learned reward model (as in RLHF), DPO operates directly on pairwise preference data — information about which of two model outputs a human prefers for a given prompt. The model is then trained to increase the likelihood of the preferred output while decreasing the likelihood of the dispreferred one.

Training Objective. DPO formalizes the learning objective as a contrastive loss between the preferred and rejected responses. The model learns a preference-consistent policy by directly optimizing the policy $\pi_\theta(y|x)$ without relying on a reward model. The objective is:

$$L_{\text{DPO}}(\theta) = -\log \sigma \left(\beta \cdot \left[\log \pi_\theta(y_{\text{preferred}}|x) - \log \pi_\theta(y_{\text{rejected}}|x) \right] \right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function, and $\beta > 0$ is a temperature parameter controlling sharpness. This loss encourages the policy to assign higher probability to preferred completions over rejected ones.

Derivation. The DPO loss is derived from a probabilistic model of human preferences. Given a pair (y^+, y^-) where y^+ is preferred over y^- for input x, we model the probability of this preference using a Bradley-Terry-style model:

$$P(y^+ \succ y^- | x) = \frac{\exp(\beta r(y^+, x))}{\exp(\beta r(y^+, x)) + \exp(\beta r(y^-, x))}$$

Here, $r(y, x)$ represents the (unknown) reward assigned to response y. Since no explicit reward model is used, we approximate $r(y, x)$ by the log-probability under the policy:

$$r(y, x) \approx \log \pi_\theta(y|x)$$

Substituting into the preference model, we obtain:

$$P(y^+ \succ y^- | x) \approx \frac{\pi_\theta(y^+|x)^\beta}{\pi_\theta(y^+|x)^\beta + \pi_\theta(y^-|x)^\beta}$$

Minimizing the negative log-likelihood of observing the preferred choice yields:

$$L_{\text{DPO}}(\theta) = -\log \left(\frac{\pi_\theta(y^+|x)^\beta}{\pi_\theta(y^+|x)^\beta + \pi_\theta(y^-|x)^\beta} \right) = -\log \sigma \left(\beta \cdot \left[\log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x) \right] \right)$$

This final form highlights the contrastive nature of DPO, which allows it to learn from preference data in a stable and scalable way without the need to train an explicit reward function.

Advantages. DPO offers several advantages over RLHF:

- **Simplicity:** Eliminates the need to separately train and maintain a reward model.
- **Stability:** Avoids the instability and tuning challenges associated with PPO or other RL algorithms.
- **Efficiency:** Requires fewer computational resources and is easier to reproduce in open-source workflows.
- **Effectiveness:** Despite its simplicity, DPO often matches or exceeds RLHF performance on alignment benchmarks.

Adoption and Applications. DPO has been rapidly adopted in the open-source community due to its lightweight design and strong empirical results. It is especially suitable for researchers, startups, and community developers who want to align models using preference data without the engineering complexity of full RLHF pipelines.

Projects like LLaMA-2-Chat and OpenChat have utilized DPO-based training to improve alignment in smaller-scale models.

Limitations. Despite its advantages, DPO also inherits some limitations of preference-based training:

- It still depends on high-quality preference data, which can be expensive to collect.
- It cannot handle reward-based supervision (e.g., correctness in math or coding) as naturally as RL methods can.
- It assumes that pairwise preferences are sufficient to guide policy improvement, which may not hold in complex reasoning tasks.

Conclusion. DPO represents a step toward making alignment more accessible and robust. By grounding learning directly in human preferences and discarding auxiliary components, it reduces overhead without sacrificing performance — aligning closely with the recent trend toward minimalist, high-leverage training techniques in LLM development.

8.2.5 Test-Time Scaling and Reasoning

While most alignment techniques focus on improving model behavior through parameter updates during training, a complementary strategy known as **test-time scaling** leverages additional computation during inference to enhance reasoning. The central insight is that advanced reasoning abilities can often be unlocked not through retraining, but by prompting the model to "think more"—that is, to generate and evaluate intermediate steps, multiple hypotheses, or structured plans before producing an answer.

Motivation. Language models can internalize vast amounts of knowledge during pretraining and posttraining. However, they do not always deploy this knowledge effectively when prompted naively. Especially in domains like mathematics, logic, code synthesis, and multi-hop reasoning, naive prompting leads to brittle or shallow outputs. Test-time reasoning methods aim to activate latent capabilities by encouraging step-by-step thinking, structured exploration, and validation before answering.

Core Techniques. Several strategies have been proposed to extend a model's reasoning abilities through inference-time methods or lightweight posttraining:

- **Chain-of-Thought (CoT)** (Wei et al., 2022): This approach instructs the model to break down its reasoning into intermediate steps before arriving at a final answer. For example, instead of asking "What is 23 + 45?", a CoT prompt might be: "Let's solve step by step: First, add the tens...". CoT significantly improves performance on arithmetic, logic, and commonsense tasks in large models.

- **Tree-of-Thought (ToT)** (Yao et al., 2023): Building on CoT, ToT enables the model to explore multiple reasoning paths in parallel. These paths can be evaluated and compared using either a value function or a separate verifier model, simulating the deliberative reasoning strategies humans often use. ToT can be viewed as a test-time Monte Carlo Tree Search (MCTS) over language.

- **Reinforcement Learning on Reasoning Tasks** (OpenAI, 2023; Bi and Team, 2024): Rather than relying on human preferences, these methods train models using verifiable correctness signals (e.g., a math solution is correct if it passes symbolic evaluation). This shifts the alignment signal from human judgments to objective outcomes, enabling more robust training for domains where correctness is unambiguous.

- **DeepSeek-R1** (Bi and Team, 2024): This is the first open-sourced reasoning-focused LLM trained entirely with reinforcement learning. It introduces a novel algorithm called Group Relative Policy Optimization (GRPO), which improves stability compared to PPO. Notably, DeepSeek-R1 eschews reward modeling and human preference data entirely, relying instead on rule-based rewards (e.g., solution accuracy, format compliance) and verification pipelines.

- **s1: Simple Test-Time Scaling** (McKenzie et al., 2025): Rather than retraining large models, s1 fine-tunes a reasoning head on a small, curated dataset (s1K) and prompts the model to allocate "compute budget" during test time to simulate deeper thinking. This method achieves notable improvements in reasoning performance at a fraction of the cost of full-scale training.

Strategic Implications. Test-time reasoning strategies are particularly valuable because they are model-agnostic: they can be applied to pretrained or posttrained models without additional parameter updates. This makes them suitable for:

- Enhancing accuracy on high-stakes domains (e.g., finance, law, education).

- Augmenting small models with structured prompting or budgeted deliberation.

- Building agentic systems that require planning, verification, or error correction.

Application in Business and Research. Test-time reasoning has practical applications in business research, such as causal inference, simulation of consumer behavior, automated agents for financial modeling, and productivity benchmarking. By structuring reasoning paths explicitly, these methods provide greater transparency and debuggability, which are essential for enterprise adoption.

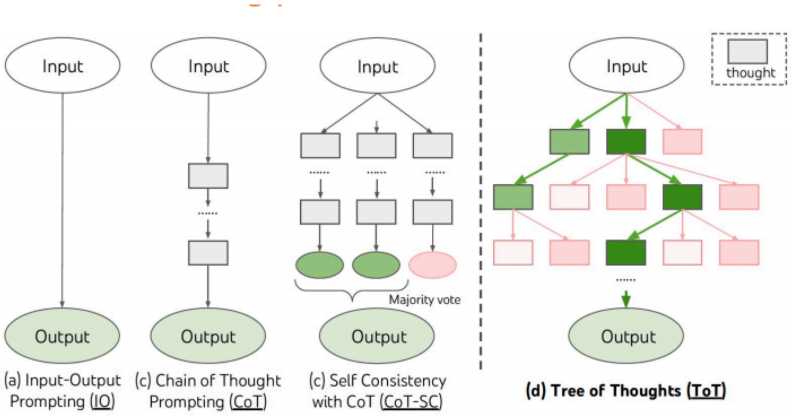

Input
Input
Input
Input
thought
Majority vote
Output
Output
Output
Output
(a) Input-Output Prompting (IO)
(c) Chain of Thought Prompting (CoT)
(c) Self Consistency with CoT (CoT-SC)
(d) Tree of Thoughts (ToT)

Figure 8.2.4. Tree-of-Thought enables structured reasoning.

8.2.6 *Knowledge Distillation (KD)*

Knowledge Distillation (KD) (Hinton et al., 2015) is a model compression technique that transfers knowledge from a large, powerful model (the *teacher*) to a smaller, more efficient model (the *student*). The goal is to retain much of the performance of the teacher while dramatically reducing the student model's size, latency, and memory footprint—making it suitable for resource-constrained settings like mobile deployment, on-device inference, or enterprise-scale usage.

Motivation. Large models such as GPT-3 and DeepSeek-V3 are costly to serve and difficult to fine-tune. KD enables deployment of lightweight variants that inherit most of the teacher's capability but are easier to operate and adapt. In this way, KD bridges the gap between model performance and deployment feasibility, especially when inference speed and hardware constraints are key concerns.

Training Objective. The student model is trained to mimic the output behavior of the teacher. This is achieved not by training on hard labels (as in standard supervised learning), but by learning to match the teacher's *soft outputs*—i.e., the probability distribution over the vocabulary. These soft labels contain rich information about class similarities and uncertainty, which can accelerate learning.

The overall objective combines two components:

- A **cross-entropy loss** with ground-truth labels (L_{CE}), to maintain task performance.

- A **KL divergence loss** with the teacher's softened outputs (L_{KL}), to learn the teacher's inductive biases and response patterns.

The total loss function is a weighted combination of the loss from both the hard labels(predicting the ground truth) and the soft labels(mimicking the teacher):

$$L_{total} = \alpha L_{CE} + (1-\alpha) L_{KL} \left(\text{softmax}\left(\frac{\text{student}}{T}\right), \text{softmax}\left(\frac{\text{teacher}}{T}\right) \right)$$

where T is a temperature parameter (typically $T > 1$) used to smooth the output distributions, and α balances the trade-off between supervised and distillation signals.

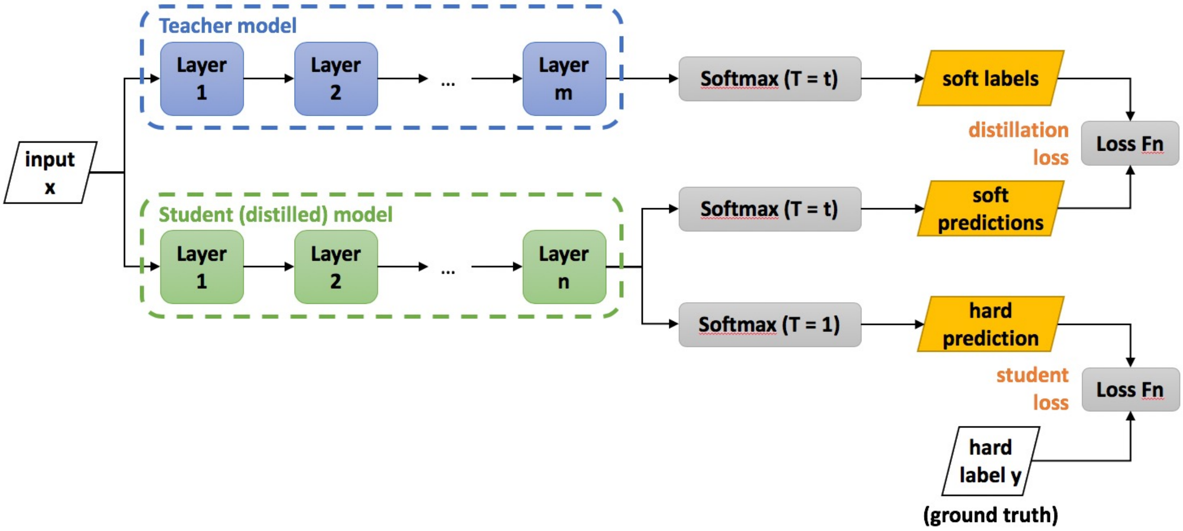


Figure 8.2.5. Knowledge distillation process from teacher to student.

Variants and Extensions. KD has evolved into several subfields:

- **Task-specific KD:** Student models fine-tuned for specific domains (e.g., biomedical, legal, financial).

- **Self-distillation:** A model distills its own knowledge into smaller variants or even into itself across training epochs.

- **Offline vs. online distillation:** In offline KD, the teacher is fixed and pre-trained; in online settings, both teacher and student may evolve during training.

Application in AI-Augmented Estimation. In applied research, KD has been used to create low-bias, low-variance estimators from LLMs. For example, Zhang et al. (2024)

introduce AI-Augmented Estimators, where a large LLM serves as a teacher to generate pseudo-labels and intermediate reasoning chains, which are then distilled into a smaller student model for efficient deployment in behavioral economics or market research tasks.

Benefits.

- **Efficiency:** Student models are significantly smaller and faster, enabling deployment on CPUs, edge devices, or real-time systems.
- **Performance retention:** When properly distilled, student models often achieve 90–95% of the teacher's accuracy on downstream tasks.
- **Fine-tuning flexibility:** Distilled models are easier to re-train or adapt to new tasks due to reduced parameter count.

Limitations.

- **Information bottleneck:** Students may fail to fully capture the nuanced behaviors of their teachers, especially in low-data regimes.
- **Teacher quality dependency:** Poorly aligned or underperforming teachers can produce low-quality soft labels, reducing distillation effectiveness.
- **Loss of interpretability:** The student may generalize differently than the teacher, complicating debugging and attribution.

Conclusion. Knowledge distillation enables scalable deployment of high-performing language models without the burdens of computational overhead. As foundation models grow larger, distillation will remain essential in bridging the research-deployment gap—empowering organizations to deliver powerful LLM-based systems with real-world constraints in mind.

8.3 The Notebooks

- The notebook LLM_FT is a demo for fine-tuning a language model using the tatsu-lab/alpaca dataset for Supervised Fine-Tuning (SFT).

9 Efficient LLM Inference

Large language models (LLMs) begin their journey with vast pretraining on web-scale corpora, followed by nuanced posttraining to align them with human intent. However, the final bottleneck for real-world deployment lies not in training—but in inference. Every downstream application, from real-time customer support to embedded agents on edge devices, hinges on the model's ability to generate outputs quickly, cost-effectively, and at scale.

Unlike training, which is typically performed once on large infrastructure, inference occurs millions or billions of times. Each token generated requires a forward pass through a subset or the entirety of the model. Thus, improving inference efficiency is not a marginal gain—it is a necessity for viability. This section explores the core strategies that enable scalable, low-latency LLM inference: **KV caching**, **quantization**, **optimized inference architectures** (e.g., DeepSeek), and **operations research (OR)** techniques for scheduling and memory-aware execution.

9.1 KV Caching: Memory as the New Compute Bottleneck

Transformer-based models generate outputs autoregressively: each token is predicted based on all prior tokens in the sequence. This means that for every new token, the model performs self-attention over an expanding sequence of inputs. In its raw form, this leads to redundant computations and exponential latency growth for long contexts.

Key Idea: Reuse Instead of Recompute. KV caching addresses this by storing the Key (K) and Value (V) vectors from previous tokens so that attention computations for subsequent tokens can reuse them. Instead of recalculating self-attention from scratch, the model:

- computes K, V for all **prompt tokens** once (prefill stage),
- appends new K, V for each generated token (decode stage),
- attends over the accumulated K, V cache with each new Query (Q).

Memory Trade-Off. While compute is reduced, memory becomes the bottleneck: naively, each token in the sequence must retain its K and V vectors across all heads and layers. As token sequences and batch sizes increase, cache memory scales linearly, quickly overwhelming even high-end GPUs. Thus, inference optimization often centers around how attention is structured—and how much cache it requires.

MLA: Latent Attention for Long-Context Models. As context windows in large language models (LLMs) continue to grow—from 4K to 128K tokens and beyond—efficient memory usage during inference becomes a critical bottleneck. The key contributor to memory usage is the **Key-Value (KV) cache**, which stores intermediate representations for every past token in the sequence. For a standard multi-head attention (MHA) mechanism, this cache grows linearly with both the number of heads and sequence length, posing serious challenges for inference scalability.

To address this, DeepSeek-V3 introduces a novel attention variant called **Multi-Head Latent Attention (MLA)** (DeepSeek-AI et al., 2025). The core idea behind MLA is to use **learned low-rank projections** to compress the Key and Value matrices for each token into a lower-dimensional latent space. Formally, for each token, instead of storing the full $K, V \in \mathbb{R}^{t_2 \times d_1}$ (where t_2 is the number of KV heads), MLA stores:

$$K' = W_K K, \quad V' = W_V V$$

where $W_K, W_V \in \mathbb{R}^{d_1 \times r}$ are learned projection matrices, and $r \ll d_1$ is a latent dimension (e.g., $r = 16$, $d_1 = 128$). The resulting $K', V' \in \mathbb{R}^{t_2 \times r}$ are much smaller and significantly reduce memory cost per token.

In addition to compressing the KV cache, MLA optionally projects the query vectors Q into the same latent space:

$$Q' = W_Q Q$$

This not only reduces memory storage but also reduces compute during the attention score calculation, since attention weights are now computed in the smaller latent space:

$$\text{Attention}(Q, K, V) \approx \text{softmax}\left(\frac{Q' K'^{\top}}{\sqrt{r}}\right) V'$$

Key Advantages of MLA:

- **Reduced KV Cache Size:** Since $r \ll d_1$, MLA dramatically lowers the per-token memory footprint, enabling support for longer contexts (e.g., 128K+ tokens) without exhausting GPU memory.
- **Multi-Head Expressiveness Retained:** Unlike MQA (which shares KV across all heads), MLA allows for per-head latent keys/values, maintaining fine-grained attention.
- **Lower Computational Cost:** When Q is also projected, the attention dot-product and value aggregation occur in a smaller space, reducing FLOPs per token.

- **Compatibility with Other Optimizations:** MLA can be combined with techniques like rotary embeddings, FlashAttention, and grouped experts.

In summary, MLA provides a principled and trainable compression mechanism for attention that balances memory efficiency with expressiveness. It is particularly well-suited for next-generation long-context LLMs that operate under strict memory constraints during inference.

KV Cache Requirements by Attention Type. During autoregressive inference with transformer models, each new token must attend to previous tokens via the self-attention mechanism. To avoid recomputing attention for all past tokens at each step, modern implementations cache the intermediate **Key (K)** and **Value (V)** tensors computed for previous tokens. This cache, known as the **KV cache**, enables efficient inference but consumes a substantial amount of memory. The size of the KV cache depends heavily on the specific attention mechanism used in the transformer.

To quantify KV cache memory cost, we define the following terms:

- t_2: the number of distinct **key/value heads**. In standard multi-head attention (MHA), this equals the number of attention heads. In shared-head mechanisms like MQA or GQA, t_2 is smaller, we denote it as t_3.
- d_1: the dimensionality of each key or value vector per attention head.
- f: the number of layers.

The total KV cache size of the Multi-head attention mechanism **per token** is proportional to $2 \cdot t_2 \cdot d_1 \cdot f$, accounting for both keys and values.

The table below summarizes how different attention mechanisms impact KV cache size and the trade-offs in terms of expressiveness and memory usage:

Attention Mechanism	**KV Cache per Token**	**Capability Profile**
Multi-Head Attention (MHA)	$2t_2d_1f$	Highest fidelity, largest cache
Grouped-Query Attention (GQA)	$2t_3d_1f$	Trade-off between capacity and cache
Multi-Query Attention (MQA)	$2d_1f$	Lightweight, lower context fidelity
Multi-Head Latent Attention (MLA)	$\approx \frac{9}{2}d_1f$	Compact and expressive (DeepSeek-V3)

Table 11. KV Cache Memory Costs across Attention Mechanisms.

Example: Suppose $d_1 = 128$ and there are two layers ($f = 2$). For 16 attention heads:

- **MHA:** $2 \cdot 16 \cdot 128 \cdot 2 = 8192$ bytes $\approx 8KB$ per token

- **MLA:** $\frac{9}{2} \cdot 128 \cdot 2 = 1152$ bytes ≈ 0.5 KB per token

This example shows that MLA requires **8× less KV cache memory per token** than standard MHA — a crucial advantage when serving long sequences or scaling up concurrent inference requests.

In summary, the choice of attention mechanism not only affects model performance but also has a significant impact on inference-time memory efficiency. Emerging models such as Mistral, LLaMA 3, and DeepSeek increasingly adopt MQA or MLA to balance capability and scalability in production settings.

9.2 Quantization: Shrinking the Model Without Shrinking Its Brain

Quantization refers to reducing the numerical precision of a model's weights and activations—from 32-bit floating-point (FP32) to formats like INT8, FP16, or even 4-bit integers. Lower precision improves memory bandwidth, increases arithmetic throughput, and allows for cheaper hardware deployment.

Linear Quantization. In linear quantization, a real-valued number r is mapped to a discrete integer q via:

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}, \quad Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

$$q = \text{round}\left(\frac{r}{S} + Z\right), \quad \hat{r} = S(q - Z)$$

Where S is the scale and Z is the zero-point offset. This transformation can be performed globally or per-channel.

Advanced Strategies.

- **Per-group/channel quantization:** Each tensor slice has its own scale/offset.
- **Quantization-aware training (QAT):** Simulates quantization during training to retain accuracy.
- **Double quantization:** Compresses quantization tables themselves.

QLoRA: Efficient Fine-Tuning with 4-Bit Models. QLoRA (Dettmers et al., 2023) combines 4-bit quantization (NF4 format) with Low-Rank Adaptation (LoRA). This enables parameter-efficient fine-tuning of massive models (e.g., LLaMA-65B) on consumer GPUs by freezing the base model and injecting small trainable matrices. The method achieves near full-precision performance while drastically reducing memory usage.

9.3 DeepSeek Inference Architecture: High-Throughput, Low-Latency Deployment

DeepSeek's inference stack is designed for production-scale operation, balancing parallelism, caching, and network latency.

System Design.

- **Hardware:** 278 nodes, each with 8 NVIDIA H800 GPUs.
- **Parallelism:** Expert Parallelism (EP) routes tokens to distinct sub-networks (MoE) across GPUs, reducing per-token load.

Performance Snapshot.

- Prefill (Input) throughput: $\sim$73.7k tokens/s
- Decode (Output) throughput: $\sim$14.8k tokens/s
- Daily input tokens: 6.088B (342B from KV cache)
- Daily output tokens: 1.688B
- Average KV-cache length: 4,989 tokens/output
- Cost: \$87,072/day vs. Revenue: \$562,027/day → 545% profit

This performance enables real-time applications—e.g., chatbots, AI copilots, or financial agents—at economically sustainable margins.

9.4 Operations Research (OR) for KV-Aware Inference Scheduling

As inference systems grow, so does the complexity of scheduling requests and allocating memory efficiently. Operations Research (OR) provides mathematical strategies to optimize system throughput and responsiveness.

The Core Challenge. When multiple prompts arrive simultaneously with varying sequence lengths, naïvely serving them in order (First Come, First Serve) leads to cache overflows and delayed responses.

Cache-Aware Scheduling. We can implement an OR-based scheduling algorithm that:

- Prioritizes prompts with the smallest expected output length.
- Preemptively clears or defers prompts that would cause KV-cache overflows.
- Outperforms FCFS and fixed-batch clearing under high concurrency.

Deployment Implications. These methods are critical in large-scale LLM deployment—ensuring that inference remains responsive under pressure and hardware is used optimally. In edge systems or real-time pipelines, such techniques directly translate into user satisfaction and operational cost reduction.

Conclusion: From Training to Serving. While model pretraining builds linguistic competence, and posttraining aligns this capability with human goals, it is inference that translates these abilities into real-world value. Efficient inference is not just an implementation detail—it is the foundation for LLM accessibility, sustainability, and commercial success. Through innovations in memory caching, numerical precision, deployment architecture, and system-level optimization, modern LLMs are not only smarter—but leaner, faster, and more affordable to run.

9.5 The Notebooks

- The notebook Data_Precision_and_Quantization illustrates how data precision and quantization impact the size of a deep learning model.

10 Research with LLMs

Large Language Models (LLMs) are no longer mere natural language generators—they are research collaborators, simulation engines, knowledge assistants, and design tools. As their capabilities scale, so too does their impact on scientific workflows. This section articulates how LLMs can be effectively integrated into research practice, examining their methodological strengths, constraints, and design considerations across empirical, computational, and interpretive domains.

10.1 Research Affordances of LLMs

LLMs augment research in four key areas:

- **Text Synthesis and Summarization:** LLMs generate human-like language, enabling the rapid drafting of literature reviews, reports, or teaching material. They also excel at compressing long documents via extractive or abstractive summarization, especially when guided with techniques like chain-of-thought prompting.
- **Ideation and Hypothesis Generation:** When prompted appropriately, LLMs can propose novel hypotheses, methodological framings, or interpretations. This aligns with cognitive offloading, where the model provides combinatorial ideation support during early-stage research.
- **Data Transformation:** With agentic extensions, LLMs can clean, reformat, and annotate datasets (e.g., tabular, textual, or code-based), reducing time-to-analysis. **Retrieval-Augmented Generation (RAG)** enhances this by grounding responses in structured corpora or enterprise databases:
  - **Indexing:** External data (e.g., research papers, corporate reports, or databases) is processed into a vector database. Documents are segmented into chunks, and each chunk is embedded using a model like BERT or Sentence-BERT, producing high-dimensional vectors that capture semantic meaning.
  - **Retrieval:** When a prompt is submitted, it is embedded into the same vector space. The system retrieves the most relevant chunks based on cosine similarity:

    $$\text{similarity} = \cos(\theta) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \|\mathbf{d}\|}$$

    where $\mathbf{q}$ is the prompt embedding and $\mathbf{d}$ is a document embedding. Typically, the top k chunks (e.g., $k = 5$) are selected to balance context and efficiency.

  - **Augmentation:** Retrieved chunks are appended to the prompt, providing context that informs the LLM's response. For example, a prompt asking for "recent trends in renewable energy" might be augmented with excerpts from 2025 industry reports.
  - **Generation:** The LLM generates a response conditioned on the augmented prompt, leveraging both its internal knowledge and the external context.

- **Simulation and Counterfactuals:** Multi-agent LLM systems (e.g., OASAS, Generative Agents) simulate complex social or economic dynamics. These simulations allow researchers to model emergent behavior, test counterfactuals, or explore narrative-based evaluation beyond standard statistical tools.

10.2 From Tool to Agent: Task-Driven Control

The leap from passive language generation to active research support demands robust *controllability*. Output quality is not just a function of architecture but of **how** the model is prompted, constrained, and interpreted. Controllability includes:

- **System prompts** are foundational instructions that define an LLM's role, tone, and constraints, ensuring consistent and task-appropriate responses. A well-crafted prompt acts as a contract between the user and the model, guiding its behavior across interactions. For example, a prompt like *"You are a PhD-level economist. Provide detailed, evidence-based analyses with citations in APA format"* sets clear expectations for scholarly output.

  Effective prompts typically include:

  - **Role Specification:** Clarifies the persona (e.g., "data scientist," "legal scholar"), aligning the model's knowledge and tone with the domain.
  - **Output Format:** Specifies structure, such as "respond in bullet points," "write a 500-word essay," or "generate Python code with comments."
  - **Constraints:** Limits undesirable behaviors, such as "avoid speculative claims," "do not use first-person pronouns," or "cite only peer-reviewed sources."
  - **Contextual Guidance:** Provides background, such as "assume familiarity with machine learning" or "focus on 21st-century economic trends."

  Prompt engineering is both an art and a science, as minor wording changes can significantly alter outputs. For instance, adding "explain step-by-step" often elicits clearer reasoning. Researchers should iteratively refine prompts, testing variations to optimize performance for specific tasks, such as drafting research proposals or summarizing complex datasets.

- **Sampling control:** Temperature, top-k, top-p, and beam search enable calibrated output diversity:
  - **Temperature (τ):** This parameter adjusts the randomness of token selection during generation. It modifies the probability distribution over the vocabulary, defined as:

    $$P(x_i) = \frac{\exp(\text{logit}(x_i)/\tau)}{\sum_j \exp(\text{logit}(x_j)/\tau)}$$

    A low temperature ($\tau < 1$, e.g., 0.2) sharpens the distribution, favoring high-probability tokens and producing deterministic, focused outputs ideal for technical writing or factual queries. A high temperature ($\tau > 1$, e.g., 1.5) flattens the distribution, encouraging diverse and creative outputs but risking incoherence. Researchers must balance temperature settings based on the task, often experimenting within the range [0.1, 2.0].
  - **Top-K Sampling:** This method restricts sampling to the K most likely tokens at each step, reducing the chance of selecting improbable tokens that could derail coherence. For example, setting $K = 50$ ensures the model samples from the top 50 candidates, maintaining quality while allowing some variability. However, a fixed K may exclude contextually relevant but less probable tokens, limiting expressiveness in creative tasks.
  - **Top-P Sampling (Nucleus Sampling):** Instead of a fixed number of tokens, Top-P sampling selects the smallest set of tokens whose cumulative probability exceeds a threshold P (e.g., $P = 0.9$). This dynamic approach adapts to the probability distribution, allowing more flexibility in high-entropy scenarios and tighter control in low-entropy ones. It often outperforms Top-K in tasks requiring nuanced language, such as literature reviews.
  - **Beam Search:** This algorithm maintains B (beam width) candidate sequences, exploring the most probable paths by maximizing the joint probability:

    $$\text{score} = \sum_{t=1}^{T} \log P(w_t | w_{1:t-1})$$

    Beam search excels in tasks requiring high coherence, such as summarization, but can produce repetitive or overly conservative outputs. A typical beam width ($B = 5$) balances exploration and efficiency, though larger values increase computational cost.
- **Planning and Tool Use:** Agent frameworks like ReAct and Reflexion integrate reasoning steps with tool calls, enabling LLMs to function as planners and API callers—not just speakers.

When integrated with toolkits such as LangChain or OpenAgents, LLMs can be embedded into end-to-end research workflows: fetching data, conducting statistical analyses, generating graphs, and interpreting results in context.

10.3 Evaluation as a Methodological Safeguard

In high-stakes research, evaluation ensures that LLM-enabled workflows are robust, reproducible, and reliable. A rigorous evaluation pipeline combines:

- **Task-Specific Metrics:** For classification, summarization, or reasoning, use precision, recall, BLEU, ROUGE, or functional correctness. In detail:
  - *Perplexity:* Historically used to measure model uncertainty, perplexity is defined as:

    $$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_{1:i-1})\right)$$

    While useful for generative models, it is less relevant for instruction-tuned LLMs, which prioritize task performance over raw language modeling.
  - *Standard Benchmarks:* Benchmarks like MMLU (Massive Multitask Language Understanding), MATH (mathematical problem-solving), and HumanEval (code generation) assess specific skills. MMLU covers 57 tasks across STEM, humanities, and professional fields, while HumanEval evaluates functional correctness of code. However, these benchmarks are vulnerable to hacking, as models can be fine-tuned to memorize answers, inflating scores without improving generalization.
  - *Human Evaluation:* Platforms like Chatbot Arena (LMSYS, 2023) facilitate head-to-head comparisons, where human judges rank model responses based on quality, relevance, or preference. While considered the gold standard, human evaluation is subjective, costly, and time-consuming, making it impractical for large-scale testing.
  - *Domain-Specific Metrics:* For fine-tuned models, custom metrics like precision, recall, or F1 score (for classification tasks) ensure alignment with research objectives. For example, a medical LLM might be evaluated on diagnostic accuracy against a curated dataset.
- **General Capability Benchmarks:** Datasets like MMLU and HumanEval provide coverage across domains, but risk overfitting. Researchers should beware of benchmark hacking.

- **Human Preference Testing:** Systems like Chatbot Arena (LMSYS, 2023) capture human judgment on response helpfulness, relevance, and style. This remains the gold standard for subjective tasks.
- **Domain-Specific Grounding:** Incorporating Retrieval-Augmented Generation or structured database integration improves factual consistency and reduces hallucination. For example, business researchers may ground financial inferences on earnings call transcripts or FRED macroeconomic databases.

Evaluation is not just about performance—it's about **alignment with research values**: clarity, verifiability, fairness, and reproducibility.

10.4 Agentic Research Workflows

Agentic LLMs allow the construction of self-reflective, self-improving research workflows. These agents combine:

- **LLM Core:** The LLM provides reasoning, language understanding, and decision-making capabilities, acting as the agent's "brain."
- **Planning:** Agents decompose complex tasks into subtasks, often using Chain-of-Thought (CoT) prompting to articulate intermediate steps. For example, an agent tasked with analyzing a dataset might plan to "load data, clean outliers, compute statistics, and visualize results."
- **Tool Use:** Agents integrate external tools, such as calculators, APIs, or code interpreters, to perform tasks beyond language generation. For instance, a research agent might query a statistical library to compute regression coefficients.
- **Reflection:** Advanced agents evaluate their actions and adjust strategies, improving performance over time. Reflection often involves self-critique or comparison against expected outcomes.

Frameworks:

- **ReAct** (Yao et al., 2022): The **Reason + Act** (ReAct) framework was designed to improve decision-making in language agents by combining two key components: *chain-of-thought (CoT) reasoning* and *tool use*. Rather than treating these as separate stages, ReAct interleaves them in a step-by-step loop: the agent first generates a reasoning trace (e.g., "I need to compute the total cost by multiplying price and quantity"), then takes an external action (e.g., calls a calculator), and then reflects on the output before continuing. This alternating sequence of thought and action enables the agent to perform complex, multi-step tasks such as math, question answering, and web navigation. The core insight is that reasoning supports better action selection, while actions provide grounded feedback to refine reasoning. ReAct significantly improves task accuracy and robustness in environments where tool usage is required.

- **Reflexion** (Shinn et al., 2023): Building on ReAct, the Reflexion framework introduces a third capability: **self-reflection**. After each episode or interaction, the agent pauses to critique its own reasoning and actions using natural language. It identifies what went wrong, summarizes lessons learned, and proposes improved strategies for future trials. For instance, if the agent fails a task due to premature decision-making, its reflection might state, ”I jumped to a conclusion without verifying all inputs. Next time I will double-check the intermediate result.” These verbal self-critiques are stored in memory and used to guide future behavior. Reflexion can be implemented as a form of ”verbal reinforcement learning” — no gradient updates are required; instead, the model adapts behavior based on internalized feedback. Empirically, Reflexion improves long-term success rates, especially in trial-and-error environments like ALFWorld or HotPotQA, where iterative correction is valuable.

  Key Differences and Contributions:

  - **ReAct**: Introduces a loop of *reasoning* → *action* → *observation*, which allows agents to incorporate tool outputs directly into their decision pipeline.
  - **Reflexion**: Adds a meta-cognitive layer, where agents reflect on prior performance, self-criticize, and revise their approach — even without access to external supervision.

  Why It Matters: Both frameworks move away from treating LLMs as one-shot black boxes. Instead, they enable more interactive and adaptive agents — capable of thinking, acting, learning from mistakes, and improving over time without fine-tuning. This aligns closely with goals in embodied AI, tool-augmented reasoning, and autonomous systems.

- **Multi-Agent Systems:** These simulate interactions among multiple LLM agents, modeling complex dynamics like collaboration or competition. Stanford’s Generative Agents (Park et al., 2023) create virtual societies where agents interact based on memory and goals, while OASAS (Lin et al., 2024) simulates organizational workflows. Such systems are valuable for studying social phenomena or optimizing team-based research processes.

In research contexts, agentic systems can:

- Conduct reproducible experiments (e.g., comparing model variants),
- Automate ablation studies,
- Build and test econometric models, or
- Simulate qualitative dialogues in policy or sociology studies.

10.5 Pitfalls and Ethical Vigilance

Though mighty as it seems to be, LLMs are fallible collaborators. Researchers must remain alert to:

- **Hallucinations:** Factual inaccuracies that persist even with grounding. Verification remains crucial.
- **Overfitting to prompts:** LLMs may become prompt-sensitive, yielding brittle results.
- **Proxy bias:** Output may reflect surface correlations rather than causal insight.
- **Synthetic data fallacy:** Using LLMs as proxies for human behavior—e.g., in focus group simulation—requires caution (Argyle et al., 2024).
- **Research ethics:** Data privacy, consent, and model transparency remain unresolved issues.

As Corrigan and Dube (Corrigan and Dube, 2024) emphasize, the promise of LLMs must be matched with econometric rigor, ethical protocols, and epistemic humility.

10.6 Conclusion: LLMs as Research Infrastructure

LLMs are not just tools—they are shaping a new research infrastructure. They introduce new ways of asking questions, gathering evidence, and generating insight. But with this transformation comes responsibility: to design prompts with care, evaluate outputs with rigor, and situate results in their epistemic context.

Used thoughtfully, LLMs can democratize research capacity, accelerate discovery, and scaffold new forms of interdisciplinary scholarship. They are not a replacement for human inquiry—but an amplifier of it.

Chapter 4: Causal Inferences and Machine Learning

In the evolving landscape of empirical research, causal inference has become a cornerstone discipline, bridging classical econometric rigor with modern machine learning innovation.

This roadmap is designed to guide a systematic study of causal inference, starting from foundational theories and extending into cutting-edge methodologies that handle complex, high-dimensional data environments.

Importantly, this roadmap introduces entirely new material incorporated into this year's course — emphasizing modern causal inference techniques that harness machine learning to strengthen identification, enhance robustness, and reveal richer causal structures. Our journey unfolds across four progressive stages:

- **Foundations of Classical Causal Inference**

  We begin with the Potential Outcomes Framework, formalizing causal questions through unobservable counterfactuals. Using the Randomized Controlled Trial (RCT) as the benchmark, we explore identification strategies in both experimental and observational settings. We develop Rubin's Causal Model, having a bite of econometricians' taste.

  Classical tools — regression adjustment, matching, propensity score weighting (IPW), and difference-in-differences (DID) — are systematically developed, grounded in assumptions like Conditional Independence (CIA) and Stable Unit Treatment Value Assumption (SUTVA).

- **Randomized Experiments and Linear Models**

  Revisiting RCTs with a *sharper statistical lens*, we analyze properties such as root-n consistency and robustness under misspecified data-generating processes.

  We transition from basic difference-in-means estimators to efficiency-improving techniques that prepare for machine learning integration.

- **Double Machine Learning (DML)**

  As datasets grow in complexity, traditional methods struggle with regularization bias and overfitting. Double Machine Learning (DML), a new core component of this year's course, addresses these challenges by combining machine learning flexibility with rigorous statistical inference.

  We build estimators that remain valid even when nuisance functions are estimated flexibly, using Neyman orthogonality and cross-fitting as key pillars.

- **Heterogeneous Treatment Effects (HTE)**

  Real-world causal effects are rarely uniform.

This new section focuses on estimating Conditional Average Treatment Effects (CATE) using methods like Causal Trees, Causal Forests, and Generalized Random Forests (GRF).

Machine learning tools for personalized treatment assignment — such as uplift modeling and meta-learning approaches — are introduced, marking a significant extension beyond average treatment effect estimation.

11 Foundations of Rubin's Causal Model

11.1 Causal Inference: From Philosophy to Scientific Methodology

Throughout history, the concept of causality—the fundamental question of *what causes what?*—has captivated human thought. Ancient philosophers, notably Aristotle, engaged with causal relationships on a philosophical level. However, it was not until the twentieth century that causality transitioned into a formal scientific discipline, largely shaped by the seminal contributions of:

- **Neyman (1923)**: Who introduced the pivotal concept of potential outcomes for the rigorous study of causal inference.
- **Rubin (1974)**: Who further developed the Rubin Causal Model (RCM), providing a comprehensive framework for designing and analyzing causal studies.

Today, causal inference offers a systematic approach to determine the independent and actual effect of interventions within intricate systems, spanning diverse fields from medical treatments to the implementation of business policies.

From Prediction to Causal Understanding. The majority of empirical analyses serve one of two primary objectives:

- **Descriptive and Predictive Analysis**: This involves summarizing existing relationships within data or forecasting future outcomes based on observed associations.

  Example: Predicting the volume of Uber rides tomorrow based on today's weather patterns and traffic conditions.

- **Causal Inference**: This focuses on investigating how specific interventions actively lead to changes in outcomes.

  Example: Estimating whether offering an Uber Ride Pass demonstrably *causes* an increase in the number of trips taken by subscribers.

Key Distinction.

- Prediction hinges on identifying observed patterns and associations within data.
- Causal inference necessitates modeling the underlying processes that generate the data and addressing counterfactual questions concerning hypothetical interventions.

The Potential Outcomes Framework To formalize our reasoning about causes and effects, we define for each unit *i*:

$$Y_i(1) = \text{The potential outcome if unit } i \text{ receives the treatment}$$

$$Y_i(0) = \text{The potential outcome if unit } i \text{ does not receive the treatment}$$

However, in reality, we only ever observe the realized outcome for each unit:

$$Y_i^{\text{obs}} = W_i Y_i(1) + (1 - W_i) Y_i(0)$$

where $W_i \in \{0, 1\}$ is an indicator variable denoting whether unit *i* received the treatment.

This fundamental constraint leads to the **Fundamental Problem of Causal Inference**:

> *For any given unit, we can never simultaneously observe both of its potential outcomes.*

Consequently, estimating causal effects invariably relies on making assumptions, employing specific study designs, or utilizing modeling approaches that allow us to infer the missing counterfactual information.

Key Identification Challenges. The pursuit of causal inference is fraught with three primary challenges:

1. **Missing Data Problem**: For each unit, we only observe one of the two potential outcomes.
2. **Confounding**: The assignment of treatment may be correlated with the potential outcomes due to unobserved factors that influence both.
3. **Selection Bias**: Individuals or units may self-select into the treatment based on characteristics that also affect the outcome of interest.

11.2 Randomized Controlled Trials (RCTs): The Gold Standard

Randomized Controlled Trials (RCTs) offer a powerful design-based solution to these challenges. By randomly assigning the treatment, RCTs aim to ensure that the potential outcomes are statistically independent of the treatment assignment:

$$(Y_i(0), Y_i(1)) \perp W_i$$

Under this crucial condition of randomization:

- The treated and control groups become comparable in expectation across all characteristics, both observed and unobserved.
- Differences in the observed outcomes between these groups provide consistent estimates of the true causal effects.

The standard method for estimating the causal effect in an RCT is the **Difference-in-Means (DM) estimator**:

$$\hat{\tau}_{\text{DM}} = \mathbb{E}[Y_i \mid W_i = 1] - \mathbb{E}[Y_i \mid W_i = 0]$$

Why RCTs are considered the Gold Standard:

- **Unbiasedness**: The Difference-in-Means estimator provides an unbiased estimate of the average treatment effect, even in finite samples.
- **Root-n consistency**: The estimator converges to the true causal effect at a rate of $\sqrt{n}$ as the sample size increases, ensuring valid asymptotic statistical inference.
- **Simplicity and Transparency**: The estimation process and the interpretation of the results are straightforward and transparent.

Despite these advantages, the practical, ethical, and logistical constraints associated with implementing RCTs, particularly in large-scale or studies involving sensitive topics, often make them infeasible.

11.3 Independence Assumptions

As we transition from the controlled environment of randomized experiments to the complexities of observational data, the identification of causal effects necessitates the adoption of stronger and more carefully justified assumptions. This section formalizes several key independence assumptions that form the bedrock of various identification strategies in causal inference.

Full Independence Assumption (IA) The **Independence Assumption (IA)** posits that the assignment of treatment is independent of both potential outcomes:

$$(Y_i(0), Y_i(1)) \perp W_i$$

Under the IA, the treatment status W_i reveals no information whatsoever about the potential outcomes $Y_i(0)$ and $Y_i(1)$. This is precisely the condition that a well-executed Randomized Controlled Trial (RCT) is designed to guarantee through the process of randomization.

When the IA holds true, the Average Treatment Effect (ATE) can be identified simply by comparing the means of the outcome variable across the treated and untreated groups:

$$\tau = \mathbb{E}[Y_i(1)] - \mathbb{E}[Y_i(0)] = \mathbb{E}[Y_i \mid W_i = 1] - \mathbb{E}[Y_i \mid W_i = 0]$$

Mean Independence Assumption (MIA) A less stringent condition is the **Mean Independence Assumption (MIA)**, which requires that the treatment status is independent of the *expectations* of the potential outcomes:

$$\mathbb{E}[Y_i(1) \mid W_i] = \mathbb{E}[Y_i(1)], \quad \mathbb{E}[Y_i(0) \mid W_i] = \mathbb{E}[Y_i(0)]$$

The MIA implies that the average values of the potential outcomes do not systematically differ between the treatment groups, even if the overall distributions of the potential outcomes might vary.
Note: The Full Independence Assumption (IA) logically implies the Mean Independence Assumption (MIA) (IA $\Rightarrow$ MIA). However, the reverse is not necessarily true; the MIA does not guarantee the IA. Thus, the MIA represents a weaker assumption than the IA.

Conditional Independence Assumption (CIA) In the realm of observational studies, where the Full Independence Assumption (IA) typically fails, researchers often invoke the **Conditional Independence Assumption (CIA)**:

$$(Y_i(0), Y_i(1)) \perp W_i \mid X_i$$

where X_i represents a set of observed pre-treatment covariates (characteristics measured before the intervention).
The CIA posits that, once we account for the observed covariates X_i, the assignment of treatment is as if it were random. In other words, conditional on these observed characteristics, there is no systematic relationship between who receives the treatment and their potential outcomes.

Under the CIA, coupled with the assumption of overlap (i.e., for all relevant values of the covariates X_i, there is a positive probability of receiving both the treatment and the control condition), the Average Treatment Effect (ATE) can be identified as:

$$\tau = \mathbb{E}_X \left[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i]\right]$$

Important: The Conditional Independence Assumption (CIA) is fundamentally untestable using the observed data alone. Its plausibility rests entirely on the researcher's judgment and the extent to which all relevant confounding factors (variables that influence both the treatment and the outcome) are observed and included in the set of covariates X_i.

Conditional Mean Independence Assumption (CMIA) A slightly less restrictive version of the CIA is the **Conditional Mean Independence Assumption (CMIA)**, which requires:

$$\mathbb{E}[Y_i(1) \mid W_i, X_i] = \mathbb{E}[Y_i(1) \mid X_i], \quad \mathbb{E}[Y_i(0) \mid W_i, X_i] = \mathbb{E}[Y_i(0) \mid X_i]$$

This implies that, conditional on the observed covariates X_i, the treatment assignment W_i does not provide any additional information about the *expected* potential outcomes.

Hierarchy of Assumptions:

The Full Independence Assumption (IA) implies the Mean Independence Assumption (MIA):

$$\text{IA} \Rightarrow \text{MIA}$$

Similarly, the Conditional Independence Assumption (CIA) implies the Conditional Mean Independence Assumption (CMIA):

$$\text{CIA} \Rightarrow \text{CMIA}$$

More generally, we have the following relationships:

$$\text{IA} \Rightarrow \text{CIA}, \quad \text{MIA} \Rightarrow \text{CMIA}$$

Summary

- The **Full Independence Assumption (IA)** represents an ideal scenario but is often unrealistic outside the context of carefully controlled randomized experiments.
- The **Conditional Independence Assumption (CIA)** is a cornerstone of causal inference in observational studies, but its validity hinges on the availability of rich and comprehensive covariate information.
- The **Mean Independence Assumption (MIA)** and the **Conditional Mean Independence Assumption (CMIA)** offer weaker but still potentially useful assumptions for achieving causal identification in specific contexts.

A thorough understanding of the subtle differences between these various independence assumptions is absolutely critical for researchers to select appropriate estimation strategies and to interpret the resulting causal inferences in a responsible and informed manner. The proof of the notices are left to the appendices.

11.4 The Rubin Causal Model: Formalizing Causal Inference

Transition to the Rubin Causal Model The independence assumptions introduced in the previous section — particularly the Independence Assumption (IA) and the Conditional Independence Assumption (CIA) — establish the conditions under which causal effects can be identified from observed data. However, they do not, by themselves, provide a complete statistical framework for conceptualizing, organizing, and estimating causal effects.

To rigorously formalize causal questions, address the missing data inherent in counterfactual outcomes, and systematically define causal estimands such as the Individual Treatment Effect (ITE), Average Treatment Effect (ATE), and related quantities, we turn to the **Rubin Causal Model (RCM)**.

The RCM situates causal inference within a missing data framework, explicitly highlighting the assumptions required for identification and clarifying the targets of estimation in both experimental and observational settings.

Basic Setup For each unit i:

- $W_i \in \{0, 1\}$ denotes treatment assignment ($W_i = 1$ if treated, $W_i = 0$ otherwise).
- $Y_i(1)$ denotes the potential outcome under treatment.
- $Y_i(0)$ denotes the potential outcome under control.

The observed outcome is:

$$Y_i^{\text{obs}} = W_i Y_i(1) + (1 - W_i) Y_i(0)$$

$$\begin{tikzcd}
 & Y_i(1) \arrow[dr, dashed, "W_i"] & \\
W_i \arrow[ur] \arrow[dr] & & Y_i^{\text{obs}} \\
 & Y_i(0) \arrow[ur, dashed, "1 - W_i"'] &
\end{tikzcd}$$

Figure 11.4.1. Switching representation of observed outcome Y_i^{obs} based on treatment W_i

The Fundamental Problem of Causal Inference For each unit, only one potential outcome is observed; the other is missing. Thus, individual-level causal effects are unobservable:

$$\tau_i = Y_i(1) - Y_i(0)$$

This missing data structure is at the heart of all causal inference challenges.

Stable Unit Treatment Value Assumption (SUTVA) RCM relies on the **Stable Unit Treatment Value Assumption (SUTVA)**, which comprises two components:

- **No interference between units**: The treatment assigned to one unit does not affect the potential outcomes of other units.
- **Consistency**: The observed outcome for a unit corresponds exactly to the potential outcome under the treatment received.

SUTVA ensures that potential outcomes are well-defined and interpretable at the individual level.

Causal Estimands Although the individual treatment effect τ_i is generally unobservable, several population-level causal parameters are of primary interest:

- **Average Treatment Effect (ATE)**:

$$\tau = \mathbb{E}[Y_i(1) - Y_i(0)]$$

- **Average Treatment Effect on the Treated (ATT)**:

$$\tau_{\text{ATT}} = \mathbb{E}[Y_i(1) - Y_i(0) \mid W_i = 1]$$

- **Average Treatment Effect on the Untreated (ATU)**:

$$\tau_{\text{ATU}} = \mathbb{E}[Y_i(1) - Y_i(0) \mid W_i = 0]$$

Each estimand answers a distinct causal question: - ATE concerns the entire population. - ATT focuses on those who actually received treatment. - ATU focuses on those who did not receive treatment.

Identification under Randomization Under the **Independence Assumption (IA)**:

$$(Y_i(0), Y_i(1)) \perp W_i$$

we can identify:

$$\tau = \mathbb{E}[Y_i(1)] - \mathbb{E}[Y_i(0)] = \mathbb{E}[Y_i \mid W_i = 1] - \mathbb{E}[Y_i \mid W_i = 0]$$

Moreover, the same expression also identifies the τ_{ATT} because in randomized experiments, the treated and untreated units are exchangeable in expectation:

$$\tau_{\text{ATT}} = \mathbb{E}[Y_i \mid W_i = 1] - \mathbb{E}[Y_i \mid W_i = 0]$$

In observational settings, we generally rely on the **Conditional Independence Assumption (CIA)**:

$$(Y_i(0), Y_i(1)) \perp W_i \mid X_i$$

to identify causal parameters by conditioning on covariates.

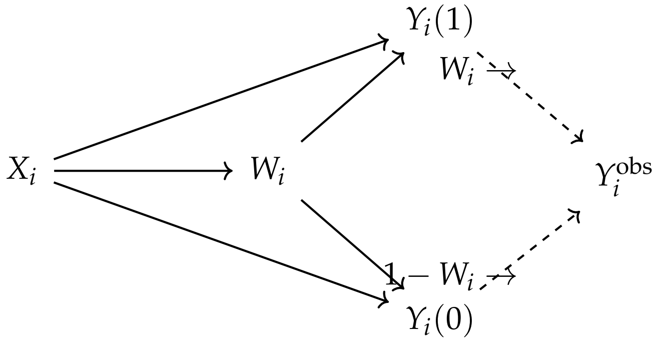


Figure 11.4.2. Switching representation of observed outcome Y_i^{obs} based on treatment W_i and Confounders X_i

Summary The Rubin Causal Model shifts causal inference into a missing data framework:

- Clearly defines causal quantities (ITE, ATE, ATT, ATU).
- Highlights the central role of assumptions (e.g., SUTVA, IA, CIA) for identification.
- Provides the foundation for developing estimation methods for causal effects in both experimental and observational settings.

Transition to Estimation Strategies While the Rubin Causal Model (RCM) provides a rigorous conceptual framework for defining and identifying causal effects, it does not in itself offer concrete estimation methods. In particular, identification of parameters such as the Average Treatment Effect (ATE) often relies on conditional expectations of observed outcomes:

$$\tau = \mathbb{E}_X \left[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}_X[Y_i \mid W_i = 0, X_i] \right]$$

Thus, the task of estimating $\mathbb{E}[Y_i \mid W_i, X_i]$ accurately and efficiently becomes central to empirical causal analysis.

One of the most fundamental and widely used approaches for this purpose is **Regression Adjustment**, which models conditional expectations directly as functions of covariates. We turn next to a systematic exploration of regression-based strategies for causal inference in observational settings.

11.5 Regression Adjustment for Causal Inference

Having established the Rubin Causal Model and the critical role of independence assumptions, we now turn to practical estimation strategies for causal effects. One of the most fundamental and widely used approaches is **Regression Adjustment**.

Regression adjustment leverages statistical modeling to estimate the conditional expectations of outcomes given treatment status and observed covariates.

Basic Idea Under the Conditional Independence Assumption (CIA):

$$(Y_i(0), Y_i(1)) \perp W_i \mid X_i$$

the average treatment effect (ATE) can be expressed as:

$$\tau = \mathbb{E}_X \left[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}_X[Y_i \mid W_i = 0, X_i] \right]$$

Thus, if we can model and estimate the conditional expectations $\mathbb{E}[Y_i \mid W_i, X_i]$ accurately, we can recover the ATE.

Linear Regression Adjustment The most straightforward approach assumes a linear relationship between outcomes, treatment, and covariates:

$$Y_i = \alpha + \tau W_i + \beta' X_i + \epsilon_i$$

where:

- τ is the coefficient on treatment, interpreted as the causal effect;

- β captures the effects of covariates X_i;
- ϵ_i is a mean-zero error term.

Ordinary Least Squares (OLS) estimation of this model provides an estimate of τ, the treatment effect.

Key Assumptions for Valid Regression Adjustment To interpret τ causally, the following conditions must hold:

- **Correct Model Specification**: The conditional expectations $\mathbb{E}[Y_i(0) \mid X_i]$ and $\mathbb{E}[Y_i(1) \mid X_i]$ are linear in X_i.
- **No Omitted Variables**: All confounders affecting both treatment assignment and outcomes must be included in X_i.
- **Common Support**: The covariate distributions in treated and control groups must sufficiently overlap.

Interpretation of Coefficients

- τ captures the difference in expected outcomes between treated and control units, adjusting for covariate differences.
- The inclusion of X_i helps correct for confounding, isolating the pure treatment effect.

Potential Issues with Regression Adjustment Despite its simplicity, regression adjustment has several potential pitfalls:

- **Model Misspecification**: If the true relationship between outcomes, treatment, and covariates is nonlinear or more complex, linear regression may yield biased estimates.
- **Extrapolation**: OLS uses a global model, which may extrapolate beyond regions with adequate data support.
- **Sensitivity to Outliers and Heteroskedasticity**: Outliers or non-constant variance can distort regression-based estimates.

Summary Regression adjustment offers a simple and powerful tool for causal estimation under CIA. However, it relies heavily on correct model specification and rich covariate information. As we will see, alternative approaches like matching and weighting seek to relax some of these modeling assumptions while achieving the same goal of adjusting for observed confounding.

11.6 Matching and Inverse Probability Weighting (IPW)

Regression adjustment relies on modeling the relationship between covariates and outcomes. An alternative approach is to **reweight** or **reconstruct** the treated and control groups to make them comparable on observed covariates, thereby mimicking a randomized experiment without imposing strong parametric modeling assumptions.

This section introduces two fundamental strategies: **Matching** and **Inverse Probability Weighting (IPW)**.

Matching: Basic Idea Under the Conditional Independence Assumption (CIA):

$$(Y_i(0), Y_i(1)) \perp W_i \mid X_i$$

the causal effect can be identified by comparing treated and control units with the same (or very similar) covariate values X_i.

Nearest Neighbor Matching (NNM) The most common matching method is **Nearest Neighbor Matching**:

- For each treated unit ($W_i = 1$), find the control unit ($W_j = 0$) whose covariates X_j are closest to X_i in terms of some distance, such as the Euclidean distance or the Mahalanobis distance.
- Estimate the treatment effect for unit i as:

$$\widehat{\tau}_i = Y_i - Y_j$$

- Aggregate across units to estimate the average treatment effect.

Propensity Score Matching (PSM) Direct matching on high-dimensional X_i can be infeasible. To simplify, we use the **propensity score**:

$$e(X_i) = \Pr(W_i = 1 \mid X_i)$$

Under CIA, it suffices to match units based on their estimated propensity scores $e(X_i)$:

$$(Y_i(0), Y_i(1)) \perp W_i \mid e(X_i)$$

Thus, **Propensity Score Matching (PSM)** proceeds by:

- Estimating $e(X_i)$ (e.g., via logistic regression).

- Matching treated and control units with similar $e(X_i)$ values.
- Estimating treatment effects by comparing matched pairs or groups.

Inverse Probability Weighting (IPW) Rather than matching units, **Inverse Probability Weighting (IPW)** reweights observations to create a synthetic sample in which treatment assignment is independent of covariates.

Each observation is weighted by the inverse of the probability of receiving the treatment it actually received:

$$\text{Weight}_i = \begin{cases} \frac{1}{\hat{e}(X_i)}, & \text{if } W_i = 1 \quad \text{(treated)} \\ \frac{1}{1-\hat{e}(X_i)}, & \text{if } W_i = 0 \quad \text{(control)} \end{cases}$$

The IPW estimator for the ATE is:

$$\hat{\tau}_{\text{IPW}} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{W_i Y_i}{\hat{e}(X_i)} - \frac{(1 - W_i) Y_i}{1 - \hat{e}(X_i)} \right)$$

Intuitively, units that are underrepresented in the treatment or control group are given more weight, balancing the sample. We would revisit this method in the next section featuring big data circumstances.

Comparison and Practical Considerations Both Matching and IPW rely on the Conditional Independence Assumption (CIA) and the propensity score. However, they differ in practical implementation:

- **Matching** emphasizes direct comparability by finding similar units.
- **IPW** adjusts the sample composition statistically through weighting.

Potential Issues:

- **Matching** can suffer from poor matches if there is limited overlap (lack of common support).
- **IPW** can suffer from high variance if propensity scores are close to 0 or 1, leading to extreme weights.

Summary Matching and IPW provide flexible, nonparametric approaches to causal inference under CIA, offering alternatives to parametric regression adjustment. Nevertheless, both require careful estimation of propensity scores and attention to overlap between treated and control populations to yield reliable causal estimates.

11.7 Structure of Modern Causal Inference: Continuity and Innovation

Modern causal inference builds fundamentally upon the Rubin Causal Model (RCM) and its associated identification strategies. It does not replace the classical framework; rather, it enhances and extends it to better address the complexities of modern empirical data.

Continuity: What Remains from the Classical Framework The core structure of causal inference remains unchanged:

- **Potential Outcomes Framework**: We still conceptualize causal questions through potential outcomes $(Y(1), Y(0))$.
- **Stable Unit Treatment Value Assumption (SUTVA)**: No interference and consistency assumptions are still necessary to define potential outcomes meaningfully.
- **Target Parameters**: Estimation of causal estimands such as the Average Treatment Effect (ATE) and Conditional Average Treatment Effect (CATE) remains the central goal.
- **Identification via Independence Assumptions**: Conditional Independence Assumption (CIA) continues to underlie the identification of causal effects from observational data.

Innovation: What is New in Modern Causal Inference While the conceptual structure persists, modern causal inference introduces critical innovations to overcome practical limitations:

- **Flexible Estimation of Nuisance Functions**: Instead of relying on simple linear regressions, we now use flexible machine learning algorithms to estimate conditional means and propensity scores, allowing for rich nonlinearities and interactions.
- **Orthogonalization and Debiasing**: To prevent overfitting biases from contaminating causal effect estimates, modern methods (e.g., Double Machine Learning) introduce orthogonal moment conditions and sample splitting strategies.
- **Heterogeneous Treatment Effects**: Classical methods focus primarily on average effects. Modern approaches directly model and estimate treatment effect heterogeneity across individuals, enabling personalized causal inference.

- **Robustness to High Dimensions**: Techniques such as cross-fitting and regularization enable valid inference even when the number of covariates is large relative to the sample size.

Summary Modern causal inference is best understood not as a break from the past, but as a natural evolution: it retains the fundamental causal structure established by the Rubin Causal Model, while augmenting estimation techniques to handle the scale, complexity, and heterogeneity inherent in contemporary data environments.

As we proceed, we will see how these innovations are systematically incorporated into frameworks such as **Double Machine Learning (DML)**, **Augmented Inverse Probability Weighting (AIPW)**, and **Causal Forests for Heterogeneous Treatment Effects (HTE)**.

In the meantime, a large portion of this section's results are mathematically-based. Along with the classic DID setting, we leave them to the appendices for reader's reference.

11.8 The Notebooks

- The notebook Causal_Unconfoundednessillustrates the performance of different propensity score based estimators under the unconfoundedness assumption.

12 Revisiting RCT with a Statistical and Big Data Taste

12.1 Motivation: Beyond the Gold Standard

As what we have mentioned before, Randomized Controlled Trials (RCTs) have long been regarded as the gold standard for causal inference, primarily because randomization ensures the independence between treatment assignment and potential outcomes, thus enabling unbiased estimation of causal effects through simple comparisons.

However, even within RCT settings, several practical and statistical challenges remain:

- **Finite Sample Uncertainty**: Although randomization guarantees unbiasedness, inference still relies on asymptotic properties, such as the Central Limit Theorem (CLT), to approximate sampling distributions and justify statistical conclusions in finite samples.
- **Efficiency Considerations**: The simple Difference-in-Means (DM) estimator, while unbiased, may be statistically inefficient. Ignoring pre-treatment covariate information leads to larger variance than necessary. Incorporating covariates can enhance precision and power.
- **Nonlinearities and Heterogeneity**: The relationship between outcomes and covariates may be nonlinear, and treatment effects may vary across individuals. Linear models may miss important heterogeneity structures, suggesting the need for more flexible modeling approaches.

Thus, modern causal inference does not aim to replace the RCT framework. Rather, it seeks to *enhance* causal estimation strategies—achieving greater efficiency, flexibility, and robustness—while maintaining the foundational logic of randomized experiments.

In what follows, we systematically explore how regression adjustment, flexible machine learning tools, and debiasing techniques can be utilized to improve causal estimation, both in randomized experiments and in observational data settings.

12.2 Statistical Inferences of RCT

In this section, we study the statistical properties of the Difference-in-Means (DM) estimator for causal effects under a randomized controlled trial (RCT) design.

Having defined the causal estimands under the potential outcomes framework, our objective is now purely statistical: to evaluate the behavior of the estimator $\hat{\tau}_{\text{DM}}$

under randomization, quantify its variability, establish its asymptotic behavior, and construct confidence intervals.

Throughout, randomness is solely induced by the random assignment of treatment.

We proceed systematically, deriving key properties: unbiasedness, variance, root-n consistency, and valid inference.

Unbiasedness of $\hat{\tau}_{\text{DM}}$ Our first goal is to establish that $\hat{\tau}_{\text{DM}}$ is an unbiased estimator for the Sample Average Treatment Effect (SATE).

This property ensures that, on average across hypothetical repetitions of the random assignment, the estimator correctly targets the causal quantity of interest without systematic error.

Theorem 12.1 (Unbiasedness of the Difference-in-Means Estimator)**.** [6]*Under SUTVA and random treatment assignment,*

$$\mathbb{E}[\hat{\tau}_{DM}] = \overline{\Delta} = \frac{1}{n}\sum_{i=1}^{n}(Y_i(1) - Y_i(0))$$

where $\overline{\Delta}$ is the Sample Average Treatment Effect.

Proof. Recall that the observed outcome is:

$$Y_i = Y_i(W_i) = W_iY_i(1) + (1 - W_i)Y_i(0)$$

and the estimator:

$$\hat{\tau}_{\text{DM}} = \frac{1}{n_1}\sum_{i=1}^{n} W_iY_i - \frac{1}{n_0}\sum_{i=1}^{n}(1 - W_i)Y_i$$

where $n_1 = \sum_{i=1}^{n} W_i$ and $n_0 = n - n_1$.

Since W_i is randomized independently of $\{Y_i(0), Y_i(1)\}$, we let

$$\mathbb{E}[W_i] = \pi, \quad \mathbb{E}[1 - W_i] = 1 - \pi$$

and we have:

$$\mathbb{E}[W_iY_i] = \pi\mathbb{E}[Y_i(1)], \quad \mathbb{E}[(1 - W_i)Y_i] = (1 - \pi)\mathbb{E}[Y_i(0)].$$

Thus:

$$\mathbb{E}[\hat{\tau}_{\text{DM}}] = \mathbb{E}[Y_i(1)] - \mathbb{E}[Y_i(0)]$$

[6] The theorem is borrowed from the Theorem 1.1 of the great book on causal inference, whose draft can be accessed via ”Causal Inference: A Statistical Learning Approach”

and in a finite sample:

$$\mathbb{E}[\hat{\tau}_{\text{DM}}] = \frac{1}{n}\sum_{i=1}^{n}(Y_i(1) - Y_i(0)) = \overline{\Delta}.$$

□

Thus, randomization guarantees that the DM estimator is centered correctly around the true sample-level causal effect.

Theorem 12.2 (Asymptotic Normality of $\hat{\tau}_{\text{DM}}$)**.** [7]*Suppose the potential outcomes* $\{Y_i(0), Y_i(1)\}$ *are i.i.d. with bounded second moments. Then:*

$$\sqrt{n}(\hat{\tau}_{DM} - \tau) \xrightarrow{d} \mathcal{N}(0, V_{DM})$$

where:

$$V_{DM} = \frac{Var[Y_i(0)]}{1-\pi} + \frac{Var[Y_i(1)]}{\pi}.$$

See Appendix B for the proof.

Theorem 12.3 (Consistency of the Plug-in Variance Estimator)**.** [8]*The plug-in estimator:*

$$\hat{V}_{DM} = \frac{n}{n_1^2}\sum_{i:W_i=1}(Y_i - \bar{Y}_1)^2 + \frac{n}{n_0^2}\sum_{i:W_i=0}(Y_i - \bar{Y}_0)^2$$

satisfies:

$$\hat{V}_{DM} \xrightarrow{p} V_{DM}.$$

See Appendix B for the proof.

Confidence Interval Construction Combining the CLT and the consistency of $\hat{V}_{\text{DM}}$, we construct asymptotically valid confidence intervals for τ.

Specifically, a $(1-\alpha)$ asymptotic confidence interval is:

$$\left[\hat{\tau}_{\text{DM}} \pm \Phi^{-1}(1-\alpha/2)\sqrt{\frac{\hat{V}_{\text{DM}}}{n}}\right]$$

where $\Phi^{-1}(\cdot)$ is the quantile function of the standard normal distribution.

As the sample size grows, the coverage probability of this interval approaches $1-\alpha$.

[6] The theorem is borrowed from the Theorem 1.1 of the great book on causal inference, whose draft can be accessed via "Causal Inference: A Statistical Learning Approach"

[7] The theorem is borrowed from the first part of the Theorem 1.2 of the great book on causal inference, whose draft can be accessed via "Causal Inference: A Statistical Learning Approach"

[8] The theorem is borrowed from the second part of the Theorem 1.2 of the great book on causal inference, whose draft can be accessed via "Causal Inference: A Statistical Learning Approach"

Through this sequence of results, we have established that the Difference-in-Means estimator:

- Is finite-sample unbiased for the Sample Average Treatment Effect (SATE);
- Is root-n consistent and asymptotically normal;
- Allows valid confidence intervals via plug-in variance estimation.

These foundational statistical properties underpin the classical use of randomized experiments in causal inference, providing both conceptual clarity and practical inferential tools.

12.3 Transition to Observational Data

In many real-world scenarios where conducting randomized experiments is not possible, researchers must rely on observational data. However, in the absence of randomization, the assignment of treatment is likely to be endogenous, meaning it is correlated with factors that also influence the outcome, thereby complicating causal interpretation.

To be specific, the elegant statistical properties derived above — unbiasedness, consistency, asymptotic normality — rest on three essential assumptions that hold in the idealized setting of a randomized controlled trial, which are dear friends we met in the last section:

1. **Independent and Identically Distributed (IID) Sampling**: Each observation $(Y_i(0), Y_i(1), W_i)$ is drawn independently from the same population distribution.
2. **Random Treatment Assignment**: The treatment W_i is assigned independently of the potential outcomes, i.e., $(Y_i(0), Y_i(1)) \perp W_i$.
3. **Stable Unit Treatment Value Assumption (SUTVA)**: The observed outcome Y_i depends only on unit i's treatment assignment W_i; there is no interference across units and no variation in treatment versions.

These assumptions form the backbone of classical causal inference. However, in modern empirical applications, particularly outside of controlled experiments, one or more of these assumptions may be violated:

- Data may exhibit dependence structures (e.g., clusters, networks), violating IID.
- Treatments may be assigned based on covariates or self-selection, undermining random assignment.

- Outcomes may be affected by others' treatment status or by different implementations of the ”same” treatment, breaking SUTVA.

Rather than abandoning inference entirely when such violations occur, a core goal of modern causal methodology is to **relax these assumptions** — one by one — in a controlled and statistically principled manner.

Statistical Relaxation Strategy We will pursue a strategy of assumption-by-assumption relaxation, in each case asking:

- What does it mean to relax this assumption?
- What new statistical structures or models are required?
- What estimators can still deliver valid inference under the relaxed setting?

This process will gradually generalize our causal inference framework, leading us from randomized experiments to observational studies, from simple samples to complex data structures, and from homogeneous treatment effects to dynamic, networked, and heterogenous causal landscapes.

Roadmap The next three subsections each focus on relaxing one classical assumption:

1. In the first, we relax the IID assumption, allowing for dependence and clustered structures. We use regression adjustment for its sake.
2. In the second, we relax the random assignment assumption, and develop estimation strategies under selection-on-observables (CIA, as noted).
3. In the third, we relax SUTVA, exploring partial interference models and sensitivity analysis.

Together, these generalizations build a bridge from the classical RCT framework to the flexible, data-rich world of modern causal inference.

12.4 Relaxing the IID Assumption: Linear and Nonlinear Specification Models

We now consider how the framework must evolve when the assumption of identically distributed observations is dropped. In realistic experimental or quasi-experimental settings, the data generating process (DGP) often exhibits heterogeneity across units: either through unit-specific variances, nonlinear responses, or unmodeled dependence structures. We begin by analyzing what fails under IID violations, and then seek robust inference strategies that remain valid under weaker conditions.

12.4.1 Linear DGP with Covariates

A key insight in modern causal inference is that randomization identifies the average treatment effect, but not all estimators are equally efficient. In particular, when potential outcomes depend on observed covariates X_i, the classical Difference-in-Means (DM) estimator may be suboptimal in terms of variance.

We now consider a more general setup where the **data-generating process** (DGP) is linear in covariates, which one should have been familiarized with in the last section.Formally, we assume that the potential outcomes are generated as

$$Y_i(w) = \alpha(w) + X_i^\top \beta(w) + \varepsilon_i(w), \quad \mathbb{E}[\varepsilon_i(w) \mid X_i] = 0, \quad \mathrm{Var}[\varepsilon_i(w) \mid X_i] = \sigma^2,$$

where $w \in \{0, 1\}$ denotes treatment status. Treatment remains randomized independently of covariates:

$$\mathbb{P}(W_i = 1) = \pi, \quad \mathbb{P}(W_i = 0) = 1 - \pi.$$

We denote $\bar{X}$ as the sample mean of the covariates, and $A = \mathrm{Var}(X_i)$ as the population variance-covariance matrix of covariates.

Instead of estimating treatment effects through simple group means, we propose running separate OLS regressions within the treated and control groups:

$$Y_i = \alpha(0) + X_i^\top \beta(0) + \varepsilon_i, \quad \text{for } W_i = 0,$$

$$Y_i = \alpha(1) + X_i^\top \beta(1) + \varepsilon_i, \quad \text{for } W_i = 1.$$

The covariate-adjusted estimator for the ATE is then defined as

$$\hat{\tau}_{\mathrm{IREG}} = \hat{\alpha}(1) - \hat{\alpha}(0) + \bar{X}^\top (\hat{\beta}(1) - \hat{\beta}(0)).$$

This estimator remains unbiased under random assignment and, crucially, achieves lower variance compared to the raw DM estimator.

Asymptotically, under mild regularity conditions, we have

$$\sqrt{n}(\hat{\tau}_{\mathrm{IREG}} - \tau) \xrightarrow{d} \mathcal{N}(0, V_{\mathrm{IREG}}),$$

where the asymptotic variance is

$$V_{\mathrm{IREG}} = 4\sigma^2 + \|\beta(1) - \beta(0)\|_A^2,$$

where we let $\|x\|_A := x' A x$ and here $A = \mathrm{Var}[X]$.

The term $\|\beta(1) - \beta(0)\|_A^2$ captures gains from controlling for covariates, reflecting how differences in covariate effects between treatment groups contribute to increased precision.

Without loss of generality, suppose that $\pi = 0.5$, we have

$$V_{\text{DM}} = \frac{\text{Var}[Y_i(0)] + \text{Var}[Y_i(1)]}{0.5} = 4\sigma^2 + \|\beta_{(0)} - \beta_{(1)}\|_A^2 + \|\beta_{(0)} + \beta_{(1)}\|_A^2$$

Comparing the adjusted and unadjusted estimators, we find

$$V_{\text{IREG}} = V_{\text{DM}} - \left(\|\beta(0)\|_A^2 + \|\beta(1)\|_A^2\right) \leq V_{\text{DM}}.$$

Thus, covariate adjustment via linear regression is strictly more efficient than the DM estimator whenever covariates have predictive power.

Finally, incorporating covariates naturally relaxes the identical distribution assumption: each unit is allowed to have its own conditional distribution indexed by X_i. Rather than assuming homogeneous outcomes across units, we exploit heterogeneity in a structured way, achieving both robustness and efficiency without sacrificing validity under randomized designs.

12.4.2 Nonlinear DGP: Randomization Without Linearity

In moving beyond the classical assumptions of identically distributed units, we now consider a more general model: the data generating process (DGP) remains randomized, but the relationship between covariates and outcomes may be arbitrarily nonlinear.

Formally, the DGP is characterized by:

$$\mu(w, x) := \mathbb{E}[Y_i(w) \mid X_i = x], \quad \sigma^2(w, x) := \mathbb{V}[Y_i(w) \mid X_i = x],$$

where $w \in \{0, 1\}$ denotes treatment status.

Treatment remains randomized independently of covariates:

$$\mathbb{P}(W_i = 1) = \pi, \quad \mathbb{E}[X_i] = 0, \quad A = \text{Var}(X_i).$$

Thus, while we retain the identification power of random assignment, we now fully accommodate nonlinear conditional expectation functions. The additional nonlinear ingredient introduced here is largely captured in the term $\mu(w, x)$, something new from it will be seen in the following paragraphs.

Variance of the Difference-in-Means Estimator Even under nonlinearity, the Difference-in-Means (DM) estimator remains unbiased for the Average Treatment Effect (ATE).

However, its asymptotic variance expands to capture additional variation driven by heterogeneity in $\mu(w, X_i)$:

$$\sqrt{n}(\hat{\tau}_{\text{DM}} - \tau) \xrightarrow{d} \mathcal{N}\left(0, V_{\text{DM}}\right),$$

where

$$V_{\text{DM}} = 4\sigma^2 + 2\,\text{Var}\left[\mu_0(X_i)\right] + 2\,\text{Var}\left[\mu_1(X_i)\right].$$

The terms $\text{Var}[\mu_w(X_i)]$ reflect heterogeneity in the regression functions across the covariate space.

Best Linear Projection Coefficients Although the true $\mu(w, X_i)$ may be nonlinear, it remains statistically advantageous to approximate it linearly.

Define the best linear projection $(\alpha^*(w), \beta^*(w))$ as:

$$(\alpha^*(w), \beta^*(w)) = \arg\min_{\alpha,\beta} \mathbb{E}\left[\left(Y_i(w) - \alpha - X_i^\top \beta\right)^2\right].$$

These projection coefficients minimize the expected squared deviation between the true conditional mean and its linear approximation.

Regression Adjustment under Nonlinear DGP We continue to use covariate-adjusted regressions separately within treatment groups:

$$Y_i = \alpha(w) + X_i^\top \beta(w) + \varepsilon_i, \quad \text{for all } i \text{ with } W_i = w.$$

The corresponding adjusted estimator for the ATE is:

$$\hat{\tau}_{\text{IREG}} = \hat{\alpha}(1) - \hat{\alpha}(0) + \bar{X}^\top(\hat{\beta}(1) - \hat{\beta}(0)).$$

Although the linear models are potentially misspecified, the estimator $\hat{\tau}_{\text{IREG}}$ remains consistent for τ under randomization.

Asymptotic Distribution of Covariate-Adjusted Estimator Under mild regularity conditions (specifically, invertibility of $\mathbb{E}[X_i X_i^\top]$), we have:

$$\sqrt{n}(\hat{\tau}_{\text{IREG}} - \tau) \xrightarrow{d} \mathcal{N}\left(0, V_{\text{IREG}}\right),$$

where

$$\begin{aligned} V_{\text{IREG}} &= \operatorname{Var}\left[X_i^\top(\beta^*(1)-\beta^*(0))\right] + \tfrac{1}{\pi}\mathbb{E}\left[\left(Y_i(1)-\alpha^*(1)-X_i^\top\beta^*(1)\right)^2\right] \\ &\quad + \tfrac{1}{1-\pi}\mathbb{E}\left[\left(Y_i(0)-\alpha^*(0)-X_i^\top\beta^*(0)\right)^2\right]. \end{aligned}$$

This variance expression captures both the systematic differences captured by the linear projections and the residual variation around those projections.

Efficiency Comparison: DM vs. IREG Even for a model where the DGP is nonlinear but still under the random treatment assignment, the IREG is more efficient than DM. A direct computation reveals that:

$$V_{\text{DM}} - V_{\text{IREG}} = \|\beta^*(0)\|_A^2 + \|\beta^*(1)\|_A^2 \geq 0,$$

Thus, regression adjustment strictly improves efficiency even when the DGP is nonlinear.

Wrap up: Power of Linear Adjustment under Nonlinearity

- Randomization protects identification of the ATE even under arbitrary nonlinearity.
- Simple Difference-in-Means estimators ignore predictable variation due to covariates, inflating variance.
- Best linear projections provide an efficient way to absorb this variation.
- Covariate adjustment remains valid and strictly more efficient than unadjusted estimation.

This benchmark nonlinear setup paves the way for further refinements: specifically, for construction of estimators that are doubly robust and orthogonal to nuisance parameter estimation.

12.5 Without Randomization: CIA-OC and Weighted IPW

In earlier sections, we relied on randomized treatment assignment to ensure that the treatment status W_i is independent of the potential outcomes $(Y_i(0), Y_i(1))$. Randomization underpins the unbiasedness and consistency of simple estimators like DM. However, in observational studies where randomization is absent, treatment decisions are often systematically related to individual characteristics. Thus, estimating causal effects requires additional, more relaxed assumptions to replace randomization at large. We come to mind of CIA to replace IA.

Together with the Overlap Condition (OC) — that $0 < \mathbb{P}(W_i = 1 \mid X_i) < 1$ — CIA ensures that we can recover the Average Treatment Effect (ATE) by appropriately adjusting for covariates.

Theorem 12.4 (Identification of ATE under CIA and OC). *Under the Conditional Independence Assumption ($\{Y_i(0), Y_i(1)\} \perp W_i \mid X_i$) and the Overlap Condition ($0 < \mathbb{P}(W_i = 1 \mid X_i) < 1$ for all X_i in the support of the covariates), the Average Treatment Effect (ATE) can be identified as:*

$$ATE = \mathbb{E}[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i]],$$

where the outer expectation is taken over the distribution of X_i.

Proof. Define the ATE as $\text{ATE} = \mathbb{E}[Y_i(1) - Y_i(0)]$. By CIA, $\{Y_i(0), Y_i(1)\} \perp W_i \mid X_i$, so:

$$\mathbb{E}[Y_i(1) \mid W_i = 1, X_i] = \mathbb{E}[Y_i(1) \mid X_i], \quad \mathbb{E}[Y_i(0) \mid W_i = 0, X_i] = \mathbb{E}[Y_i(0) \mid X_i].$$

The conditional treatment effect is:

$$\mathbb{E}[Y_i(1) - Y_i(0) \mid X_i] = \mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i].$$

Taking the expectation over X_i:

$$\text{ATE} = \mathbb{E}[\mathbb{E}[Y_i(1) - Y_i(0) \mid X_i]] = \mathbb{E}[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i]].$$

The Overlap Condition ensures that $0 < \mathbb{P}(W_i = 1 \mid X_i) < 1$, so both conditional expectations are well-defined for all X_i. Thus, the ATE is identifiable. □

Without CIA, no amount of weighting, matching, or stratification can credibly estimate causal effects.

Corollary 12.4.1 (Failure of ATE Estimation without CIA). *If the Conditional Independence Assumption ($\{Y_i(0), Y_i(1)\} \perp W_i \mid X_i$) does not hold, then no adjustment method (e.g., weighting, matching, or stratification on X_i) can consistently estimate the ATE, as there exist unmeasured confounders that bias the treatment effect estimate.*

Remark (Intuitive Explanation for CIA Violation). *Imagine evaluating a job training program's effect on earnings, using data on age, education, and experience. If personal motivation, which isn't measured, drives both program participation and higher earnings, your analysis might overestimate the program's effect. Motivated people join the program and earn* *more anyway, but without data on motivation, you can't separate its influence from the program's true impact. It's like judging a chef's dish without knowing some ingredients were pre-seasoned—you're misled by a hidden factor. No adjustment method can fix this without measuring motivation.*

The similar results hold for the case where the OC is violated.

Corollary 12.4.2 (Failure of ATE Estimation without OC)**.** *If the Overlap Condition* ($0 < \mathbb{P}(W_i = 1 \mid X_i) < 1$) *is violated, such that* $\mathbb{P}(W_i = 1 \mid X_i) = 0$ *or* $\mathbb{P}(W_i = 1 \mid X_i) = 1$ *for some* X_i*, then the ATE cannot be consistently estimated for the entire population, as no data exist for some covariate values under one of the treatment conditions.*

Remark (Intuitive Explanation for OC Violation)**.** *Consider a study on a heart medication's effect on recovery, with age as a covariate. If doctors never prescribe the drug to patients over 80, you have no data on how it affects them. Estimating the drug's effect across all ages is like guessing how a car drives on an untested road—you can't know without data. Methods like matching or weighting fail because there's no treated group over 80 to compare, leaving you unable to estimate the effect for the whole population.*

Stratified Estimator: Local Balancing via Grouping One intuitive approach to leverage the Conditional Independence Assumption (CIA) is **stratification**: grouping units into discrete strata where units have similar covariate profiles or estimated propensity scores.

Formally, partition the sample into strata $s = 1, \ldots, S$ based on X_i or $\hat{e}(X_i)$.

Within each stratum s, compute average outcomes for treated and control units:

$$\hat{\mu}_1(s) = \frac{1}{n_1(s)} \sum_{i \in s, W_i = 1} Y_i, \quad \hat{\mu}_0(s) = \frac{1}{n_0(s)} \sum_{i \in s, W_i = 0} Y_i.$$

Aggregating across strata yields the stratified estimator:

$$\hat{\tau}_{\text{strat}} = \sum_{s=1}^{S} \frac{n(s)}{n} \left(\hat{\mu}_1(s) - \hat{\mu}_0(s) \right),$$

where $n(s)$ is the number of units in stratum s.

The intuition is that within each stratum, treatment assignment is approximately random, enabling locally unbiased comparisons.

Similarly, the stratified estimator is also root-n consistent. This result holds under the CIA, SUTVA, OC and some technical conditions. Specifically, denote $V_{\text{strat}} := \frac{\sigma_1^2(x)}{e(x)} + \frac{\sigma_0^2(x)}{1-e(x)}$, we have

$$\sqrt{n(s)}(\hat{\tau}_{\text{strat}}(x) - \tau(x)) \xrightarrow{d} \mathcal{N}(0, V_{\text{strat}}),$$

where $\sigma_w^2(x) := \mathrm{Var}[Y_i(w)|X_i = x]$, $e(x) := \mathbb{P}[W_i = 1|X_i = x]$ and $\tau(x) = \mathbb{E}[Y_i(1) - Y_i(0)|X_i = x]$.

Inverse Propensity Weighting (IPW): Continuous Balancing via Reweighting Note that stratified estimator is just propensity score estimation on a discrete set. When the covariates are continuous, instead of grouping, we can reweight each observation individually according to its estimated propensity score. This is commonly used in econometrics, as noted in the introduction to PSM.

The normalized IPW estimator is given by:

$$\hat{\tau}_{\text{nIPW}} = \frac{\sum_{i=1}^{n} \frac{W_i Y_i}{\hat{e}(X_i)}}{\sum_{i=1}^{n} \frac{W_i}{\hat{e}(X_i)}} - \frac{\sum_{i=1}^{n} \frac{(1-W_i)Y_i}{1-\hat{e}(X_i)}}{\sum_{i=1}^{n} \frac{1-W_i}{1-\hat{e}(X_i)}}.$$

Each unit's contribution is scaled inversely by the estimated probability of receiving the treatment actually assigned. Normalization ensures that the weighted sample size is properly scaled, controlling for instability when $\hat{e}(X_i)$ is close to 0 or 1.

Connection between Stratification and Normalized IPW At a deeper level, stratification and normalized IPW both aim to restore the balance that randomization would have guaranteed.

Feature	Stratified Estimator	Normalized IPW
Level of balancing	Group-level (coarse)	Unit-level (fine-grained)
Mechanism	Compare means within bins	Reweight each unit individually
Dependence	On stratum definition	On estimated propensity score

As the number of strata increases (i.e., strata become finer), the stratified estimator asymptotically approximates normalized IPW. Thus, normalized IPW can be interpreted as a form of *continuous stratification* — balancing covariates across the entire support of X_i.

12.5.1 The Limitations of IPW and the Emergence of Balancing Weights

IPW provides a powerful mechanism to achieve covariate balance in observational studies under CIA and OC. However, IPW estimators face critical limitations that motivate further methodological development:

Suppose the true propensity score $e(X)$ were known. Then, the IPW estimator would be semiparametrically efficient, achieving root-n consistency, specifically,

Theorem 12.5. [9]*Suppose that $\{X_i, Y_i(0), Y_i(1), W_i\}_{i=1}^n \overset{iid}{\sim} P$, that CIA, OC and SUTVA hold, and that all moments used in the expression for V_{IPW^*} below are finite. Then, the oracle IPW estimator is unbiased, $\mathbb{E}[\hat{\tau}_{IPW^*}] = \tau$, and*

$$\sqrt{n}(\hat{\tau}_{IPW^*} - \tau) \to N(0, V_{IPW^*}).$$

$$\begin{aligned} V_{IPW^*} = \operatorname{Var}[\tau(X_i)] + \mathbb{E}\left[\frac{(\mu_{(0)}(X_i) + (1 - e(X_i))\tau(X_i))^2}{e(X_i)(1 - e(X_i))}\right] \\ + \mathbb{E}\left[\frac{\sigma^2_{(1)}(X_i)}{e(X_i)} + \frac{\sigma^2_{(0)}(X_i)}{1 - e(X_i)}\right]. \end{aligned}$$

Remark. *Note from the theorem 12.5, $V_{IPW^*} = V_{strat} + \mathbb{E}\left[\frac{(\mu_{(0)}(X_i)+(1-e(X_i))\tau(X_i))^2}{e(X_i)(1-e(X_i))}\right]$, which means that the stratified estimator is more efficient than the IPW estimator. Hence, when the covariate X is discrete with a natural but specific propensity model, one feasible IPW can outperform the oracle IPW.*

In practice, we don't have an oracle to calculate the $e(X)$, which must be estimated from data, yielding an approximation $\hat{e}(X)$. This induces additional bias and variance in the IPW estimator.

Formally, we can decompose the estimation error:

$$\hat{\tau}_{\text{IPW}} - \tau = \underbrace{(\hat{\tau}^{\text{oracle}}_{\text{IPW}} - \tau)}_{\text{oracle IPW error}} + \underbrace{(\hat{\tau}_{\text{IPW}} - \hat{\tau}^{\text{oracle}}_{\text{IPW}})}_{\text{error due to } \hat{e}(X) \text{ estimation}}.$$

Bounding the second term using Cauchy-Schwarz inequality yields:

$$\left|\hat{\tau}_{\text{IPW}} - \hat{\tau}^{\text{oracle}}_{\text{IPW}}\right| \lesssim \sqrt{\mathbb{E}[Y_i^2]} \times \sqrt{\mathbb{E}\left[(\hat{e}(X_i) - e(X_i))^2\right]}.$$

Thus, the root-mean-squared error (RMSE) of $\hat{e}(X)$ directly impacts the performance of IPW.

In finite samples:

- If $\hat{e}(X)$ is estimated at parametric rates ($n^{-1/2}$), the second term becomes asymptotically negligible.
- However, flexible machine learning models often achieve slower rates (e.g., $n^{-1/4}$), making this bias non-negligible.

[9] The theorem is borrowed from the Theorem 2.2 of the great book on causal inference, whose draft can be accessed via "Causal Inference: A Statistical Learning Approach"

That is, in finite samples, plug-in IPW estimators often have inflated bias and variance, and do not achieve semiparametric efficiency.

The Importance and Fragility of OC The performance of IPW critically depends on the validity of the Overlap Condition:

$$\eta \leq e(X_i) \leq 1 - \eta \quad \text{for all } X_i \in \mathcal{X},$$

where $\eta > 0$ is a positive constant.

When overlap is weak or fails:

- Propensity scores approach 0 or 1, leading to extreme inverse weights.
- A few observations dominate, causing instability and inflated variance.

Consequences of Poor Overlap:

- Regression adjustment becomes sensitive to extreme covariates.
- Matching may discard many units due to lack of comparable counterparts.
- Stratification suffers residual imbalance even within strata.

Practical Remedies:

- **Trimming:** Exclude units with estimated propensity scores outside $[\alpha, 1 - \alpha]$.
- **Winsorization:** Cap propensity scores at $[\alpha, 1 - \alpha]$.
- **Conditional ATE Estimation:** Restrict estimation to the subpopulation with sufficient overlap.

Trimming trades off generalizability for improved statistical stability.

IPW as a Special Case of Balancing Weights Recognizing the fragility of simple IPW estimators leads to a broader conceptual framework: **IPW is a special case of balancing weights**.

More generally:

- Assume the observed covariates X have density $f(x)$ with respect to some measure μ.
- Suppose we wish to reweight the sample to target a different population with density $g(x)$.

Define the **tilting function**:

$$h(x) = \frac{g(x)}{f(x)},$$

which reweights the observed sample to represent the target population.

The reweighted ATE is:

$$\tau_h = \mathbb{E}_g \left[Y_i(1) - Y_i(0) \right] = \frac{\int (Y_i(1) - Y_i(0)) h(x) \mu(dx)}{\int h(x) \mu(dx)}.$$

Thus:

- Standard IPW corresponds to $g(x) = f(x)$ (i.e., targeting the observed sample distribution).
- Trimming and other overlap adjustments modify $g(x)$ by restricting support.

This general balancing weights framework allows:

- Tailoring the estimand to specific policy-relevant subpopulations,
- Reducing variance through optimal weighting,
- Mitigating instability from poor overlap.

Summary Finite-sample inefficiencies, fragile overlap, and rigid weighting motivate a transition from simple IPW to more flexible, robust balancing approaches.

The balancing weights perspective generalizes IPW, providing a foundation for robust, efficient causal effect estimation.

12.6 AIPW and Double Robustness

The limitations of plug-in IPW estimators and the instability induced by poor overlap motivate the search for more robust causal estimators. One of the most influential developments in this direction is the **Augmented Inverse Propensity Weighting (AIPW)** estimator, which combines propensity score weighting and outcome regression adjustment into a unified framework.

The AIPW estimator belongs to the broader class of *doubly robust* estimators — estimators that achieve consistency if either the propensity score model or the outcome model is correctly specified, but not necessarily both.

Definition and Intuition Given $\hat{e}(X_i)$, the estimated propensity score and $\hat{\mu}_w(X_i)$, the estimated conditional mean outcome for treatment $w \in \{0,1\}$, the AIPW estimator for the Average Treatment Effect (ATE) is defined as:

$$\hat{\tau}_{\text{AIPW}} = \frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{W_i}{\hat{e}(X_i)} - \frac{1-W_i}{1-\hat{e}(X_i)}\right)(Y_i - \hat{\mu}_{W_i}(X_i)) + \hat{\mu}_1(X_i) - \hat{\mu}_0(X_i)\right].$$

The structure of AIPW reflects two adjustment channels:

- The first term uses inverse propensity weighting to balance residuals (outcome deviations from model predictions).
- The second term adjusts differences in modeled potential outcomes directly.

This construction ensures that even if one model is misspecified, the other can "correct" the bias, offering a powerful layer of protection.

Double Robustness: Why It Matters The **double robustness property** states that if either the outcome model $\hat{\mu}_w(X)$ is correctly specified, or the propensity score model $\hat{e}(X)$ is correctly specified (but not necessarily both), then $\hat{\tau}_{\text{AIPW}}$ remains consistent for τ.

This offers an empirical advantage:

- Researchers often do not know which model — the treatment assignment or the outcome process — is easier to approximate.
- Double robustness allows them to hedge against potential model misspecification.

Moreover, if **both** models are correctly specified, the AIPW estimator attains the **semiparametric efficiency bound** — achieving the lowest possible asymptotic variance among regular estimators.

Hidden Assumptions: Potential Violations of SUTVA While AIPW offers robustness against model misspecification, it still relies fundamentally on key assumptions from the Rubin Causal Model (RCM), particularly the **Stable Unit Treatment Value Assumption (SUTVA)**.

Recall that SUTVA includes two components:

- **No interference:** One unit's treatment assignment does not affect another unit's potential outcomes.

- **Consistency:** The potential outcome under treatment assignment $W_i = w$ corresponds exactly to the observed outcome when $W_i = w$.

In applying AIPW, violations of SUTVA may occur subtly but importantly:

- **Violation of No Interference:** In settings with network interactions, spillovers, or contagion effects (e.g., social networks, public health interventions), an individual's outcome may depend on other individuals' treatment statuses.
- **Violation of Consistency:** If treatments are not consistently defined across units — for instance, heterogeneous implementation of "the same" intervention — then $Y_i(W_i)$ may not represent a stable, well-defined potential outcome.

Thus, while AIPW protects against *modeling risks*, it does not protect against *structural violations* of the causal framework itself. When SUTVA fails, even a correctly specified propensity score or outcome model may not recover valid causal estimates.

Summary

- AIPW achieves double robustness by combining outcome regression and propensity weighting.
- It remains consistent if either model is correctly specified and is semiparametrically efficient if both are correct.
- However, AIPW still relies critically on foundational causal assumptions like SUTVA.
- Empirical applications must carefully assess whether interference, heterogeneous treatments, or other violations of SUTVA may threaten identification.

Understanding these structural assumptions is crucial before applying AIPW in practice.

12.7 The Notebooks

- The notebook IPW_vs_AIPW illustrates the comparison between IPW and AIPW to estimate the ATE under the unconfoundedness and overlapping conditions.

13 Double Machine Learning

13.1 From Classical Designs to Modern Data Environments

In the preceding sections, we have revisited the power of randomized controlled trials (RCTs) and explored how modern causal inference builds upon this foundation through the Rubin Causal Model (RCM), regression adjustment, inverse probability weighting (IPW), and doubly robust estimators such as AIPW.

These tools offer considerable robustness and transparency under structured designs and moderate-dimensional covariate spaces. However, contemporary data environments increasingly challenge the assumptions that underpin classical methods. In many real-world applications:

- The covariate space X is high-dimensional or even infinite-dimensional (e.g., text, image, or behavioral traces),
- Treatment assignment may depend on complex, nonlinear interactions that defy parametric modeling,
- The overlap condition(OC) may only hold in sparse or irregular regions of covariate support.

The intersection of machine learning (ML) and causal inference has therefore garnered growing attention. Yet, applying ML tools directly to causal problems is far from straightforward. As noted by Athey and Imbens (2016), *cross-validation* — a standard technique for hyperparameter tuning — cannot be reliably applied to causal estimands due to the fundamental unobservability of counterfactuals.

Moreover, strong predictive performance on propensity scores or potential outcomes does not imply valid estimation of causal effects. As emphasized by Belloni et al. (2014, 2016), regularized ML methods, such as Lasso, may introduce additional bias if post-selection inference is not handled properly, especially in high-dimensional settings.

These concerns are compounded by longstanding identification challenges in causal inference, including unmeasured confounding, lack of common support, and covariate imbalance. Crucially, ML cannot *magically solve* these problems — rather, it must be carefully integrated within causal inference frameworks.

In response, the **Double Machine Learning (DML)** framework has emerged as a principled solution. DML blends the flexibility of modern machine learning with formal guarantees from semiparametric theory. Specifically, it enables valid inference on causal parameters in the presence of high-dimensional covariates and complex nuisance structures by constructing orthogonal (or "locally insensitive") estimating equations that mitigate the impact of first-stage errors (Belloni et al., 2014, 2016).

Compared to other domains of business and social science research, this remains a highly active and rapidly evolving area of methodological innovation.

In these high-dimensional and data-adaptive environments, two central questions arise:

1. How can we estimate nuisance components — such as outcome regressions $\mu_w(X)$ and propensity scores $e(X)$ — flexibly using ML methods?

2. How can we ensure that causal effect estimators remain statistically valid, with root-n consistency and valid confidence intervals, even when ML methods introduce regularization bias or complex overfitting patterns?

These questions motivate a shift from traditional plug-in estimators toward frameworks that combine flexible function estimation with statistical orthogonality and debiasing.

Double Machine Learning (DML) offers a general framework for estimating treatment effects using machine learning methods, by leveraging the idea of *Neyman orthogonality*.

In classical causal inference, consistent estimation typically requires modeling conditional expectations such as the outcome regression $\mu_w(X) = \mathbb{E}[Y \mid W = w, X]$ and the propensity score $e(X) = \mathbb{P}[W = 1 \mid X]$. Traditional econometric approaches rely on strong parametric assumptions — such as linearity or additive separability — which, if misspecified, can lead to substantial bias.

By contrast, DML frameworks allow these functions to be learned flexibly from data using high-capacity machine learning algorithms. In doing so, DML sidesteps the need for strict functional form assumptions, while still enabling valid inference.

The DML approach is grounded in three key principles:

- **Orthogonal Estimating Equations:** The causal parameter (e.g., the ATE or the treatment effect in a partially linear model) is defined via a moment condition that is *locally insensitive* to small errors in the estimated nuisance functions. Known as *Neyman orthogonality*, it ensures that errors in $\hat{e}(X)$ or $\hat{\mu}_w(X)$ only affect the estimator's variance — not its bias — to first order.

- **Sample Splitting and Cross-Fitting:** To avoid overfitting, DML splits the sample into folds. Nuisance parameters are estimated on one fold and then used to construct orthogonal scores on a different fold. This procedure ensures the independence between the estimated nuisance functions and the target moment equations.

- **Asymptotic Validity Under Weak Rates:** Unlike classical methods, which often require $n^{-1/2}$-rate convergence of all components, DML only requires the product of nuisance function errors to converge faster than $n^{-1/2}$. In particular, DML remains valid when the nuisance functions converge at rates as slow as $n^{-1/4}$ in mean squared error (MSE) norm — a common scenario for flexible nonparametric or ML models.

Formally, DML requires:

- Satisfying some regularity conditions analogous to those in AIPW (e.g., overlap, SUTVA).
- That the ML estimators of the nuisance functions converge in L_2 or RMSE norm at a rate of $o(n^{-1/4})$, slower than the parametric $o(n^{-1/2})$ rate, but sufficient to preserve root-n consistency for the target parameter via orthogonalization.

What does DML deliver?

- *Root-n consistent* estimators of treatment effects — converging to the true value at rate $O(n^{-1/2})$.
- In frequentist terms, this implies that the estimator is *asymptotically normal*, allowing us to construct valid confidence intervals and conduct inference, even when ML-based nuisance functions are used.

This yields a powerful and generalizable approach to causal inference under complex and high-dimensional data structures.

We then turn to a concrete instance of the DML framework in the main part of this section:

- On **Partial Linear Model**(PLM): This is built on the seminal work of Robinson (1988) which introduces root-n consistent estimation in semiparametric models;
- Then we have a birdview on the DML framework which formalizes this using *Neyman orthogonality* and sample-splitting techniques(Chernozhukov et al., 2018);
- Lastly, we would go over a recent work(Farrell et al., 2021) which further extends this approach using deep neural networks for inference under high complexity.

13.2 Partial Linear Model

13.2.1 Impact of Confounders on Causal Effect Identification

To illustrate the core ideas of the Double Machine Learning (DML) framework, we adopt the **Partially Linear Model (PLM)** as a canonical example. This semiparametric model helps clarify:

- Why machine learning is essential in flexibly estimating nuisance functions;
- How statistical theory ensures valid inference even under complex data-generating processes.

Motivation. The presence of *unobserved confounding variables* fundamentally complicates the identification of causal effects. Without further assumptions — such as instrumental variables or structural modeling — causal effects cannot be point-identified in the presence of unobserved confounding.

To circumvent this, the PLM framework assumes *selection on observables*: all relevant confounders are captured by the observed covariates X. Under this condition, we can proceed to estimate causal effects using regression-based adjustments.

Model Specification. The PLM assumes the following structural equations:

$$Y = D\theta_0 + g_0(X) + U,$$

$$D = m_0(X) + V,$$

where:

- Y is the observed outcome;
- D is the treatment variable (binary or continuous);
- $X \in \mathbb{R}^p$ denotes a vector of observed covariates (measured confounders);
- $g_0(X)$ is a nonparametric function capturing how X affects Y;
- $m_0(X)$ captures the conditional expectation of D given X;
- U and V are mean-zero unobserved error terms;
- X can be high-dimensional, which means that $\dim(X) > \dim(D)$.

Assumptions. The key identification assumption is the *conditional exogeneity* of the treatment and outcome errors:

$$\mathbb{E}[U \mid X, D] = 0 \quad \text{and} \quad \mathbb{E}[V \mid X] = 0.$$

These conditions imply:

- There is no unobserved confounding after conditioning on X;
- D is as-good-as-randomly assigned, conditional on X;
- The treatment effect θ_0 can be interpreted causally under this ignorability condition.

The parameter θ_0 represents the *causal effect of D on Y*, net of the influence of covariates X. Our objective is to estimate θ_0 accurately and efficiently, even when $g_0(X)$ and $m_0(X)$ are complex or high-dimensional — a challenge addressed using DML techniques.

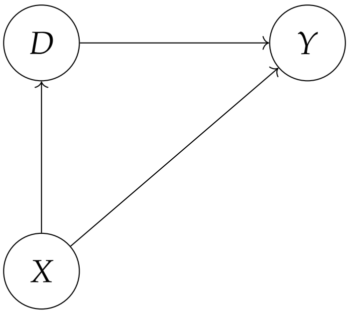


Figure 13.2.1. Partially Linear Model (PLM) Causal Diagram. The outcome Y depends linearly on treatment D and nonparametrically on covariates X. Covariates X also influence the treatment variable D. More complicated version see Chernozhukov et al. (2024a).

13.2.2 *Neyman Orthogonality: A Pillar of Double Machine Learning*

A central innovation in the Double Machine Learning (DML) framework is the use of **Neyman orthogonality** to mitigate the impact of regularization and overfitting biases when estimating treatment effects in the presence of high-dimensional nuisance parameters (Chernozhukov et al., 2018; Belloni et al., 2014).

In the context of partial linear models (PLM), recall our objective is to estimate a low-dimensional parameter θ_0 in the presence of complex nuisance functions $\eta_0 = (g_0, m_0)$, where:

$$Y = D\theta_0 + g_0(X) + U, \quad D = m_0(X) + V,$$

with zero conditional mean assumptions:

$$\mathbb{E}[U \mid X, D] = 0, \quad \mathbb{E}[V \mid X] = 0.$$

The Neyman Orthogonality Condition. To ensure robustness of $\hat{\theta}_0$ to small estimation errors in the nuisance components $g_0(X)$ and $m_0(X)$, DML constructs estimators based on orthogonal (or locally insensitive) moment conditions. Specifically, define the score function:

$$\psi(W; \theta, \eta) = (D - m(X)) \cdot (Y - g(X) - (D - m(X))\,\theta),$$

where $W = (Y, D, X)$ and $\eta = (g, m)$ are nuisance functions.

The Neyman orthogonality condition requires that the Gateaux derivative of the score with respect to η vanishes at the true nuisance parameter η_0:

$$\partial_\eta \mathbb{E}\left[\psi(W; \theta_0, \eta)\right]\Big|_{\eta=\eta_0} = 0. \tag{29}$$

This condition ensures that $\psi(W; \theta_0, \eta)$ is *locally insensitive* to first-order errors in the estimation of η_0. As a result, the plug-in estimator of θ_0 remains robust to small perturbations in $\hat{g}(X)$ and $\hat{m}(X)$.

Why Orthogonality Matters. In high-dimensional or nonparametric settings, nuisance estimators typically converge at slower rates (e.g., $n^{-1/4}$). Without orthogonality, such convergence would contaminate the consistency and efficiency of the target parameter θ_0. Neyman orthogonality guarantees that the slower convergence of $\hat{\eta}$ does not inflate the asymptotic bias of $\hat{\theta}_0$.

Thus, under mild regularity conditions:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) \xrightarrow{d} \mathcal{N}(0, \sigma^2),$$

even when $\hat{g}(X)$ and $\hat{m}(X)$ are estimated using machine learning algorithms.

This orthogonality-based debiasing approach was first formalized in classical statistics by Neyman (1959) and has been extensively developed for high-dimensional econometrics in Chernozhukov et al. (2018).

Connection to Residual-on-Residual Regression. This orthogonality condition also underlies the residual-on-residual regression representation commonly used in DML.

In this formulation, $\hat{V}_i = D_i - \hat{m}(X_i)$ and $\hat{U}_i = Y_i - \hat{g}(X_i)$ are used to estimate:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i=1}^{n} \hat{V}_i^2 \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} \hat{V}_i \hat{U}_i \right),$$

which enjoys orthogonal score construction by design.

Neyman orthogonality thus serves as the foundation for achieving valid statistical inference in the high-dimensional ML-integrated causal inference environment. Due to the core functionality it serves in the modern causal inference advancement, we will frequently see the concept in the following paragraphs.

13.2.3 Why Machine Learning Alone Is Not Sufficient in PLM

Although the above specification of Partially Linear Model (PLM) assumes that all confounders X are observed (i.e., CIA holds), this does not mean that we can naively use machine learning (ML) to estimate the causal parameter θ_0. Even if we accurately estimate the nuisance functions $g_0(X)$ and $m_0(X)$, this alone does not guarantee valid inference for θ_0.

As noted by Chernozhukov et al. (2024a), ML-based plug-in methods encounter two primary sources of bias:

- 1. Regularization Bias. ML models typically use regularization to avoid overfitting, but this induces bias in the estimated functions $\hat{g}_0(X)$ and $\hat{m}_0(X)$. If this bias is not appropriately corrected, it will contaminate the estimation of θ_0, leading to inconsistent or asymptotically biased estimators. Even popular regularized methods like Lasso suffer from this issue if post-selection bias is not addressed (Belloni et al., 2016).

- 2. Overfitting Bias. ML models often overfit the data used to train them. When the same data is used both to estimate $\hat{g}_0(X)$ and to compute $\hat{\theta}_0$, residuals will be correlated with the estimated nuisance functions. This violates orthogonality and inflates the variance of $\hat{\theta}_0$, undermining root-n consistency.

Naïve Plug-in Estimation Fails A seemingly natural approach is to fit $\hat{g}_0(X)$ using an ML method, and then plug this into a linear residual regression:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i=1}^{n} D_i^2 \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} D_i (Y_i - \hat{g}_0(X_i)) \right).$$

However, this estimator is generally not root-n consistent. The estimation error from $\hat{g}_0(X)$ enters the score function directly, and since the moment condition is not orthogonal, even small errors in $\hat{g}_0(X)$ can bias $\hat{\theta}_0$. Formally:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) \overset{d}{\not\to} \mathcal{N}(0, \cdot).$$

Double Machine Learning as a Solution **Double Machine Learning (DML)** addresses these issues using two key innovations:

- *Neyman Orthogonality.* Instead of using a standard plug-in score, DML constructs an orthogonal moment condition—one that is **locally insensitive** to first-order perturbations in the nuisance parameters. For example, the score function in an AIPW-style estimator takes the form:

  $$\psi(W_i; \theta, \eta) = (D_i - \hat{m}_0(X_i)) \cdot (Y_i - \hat{g}_0(X_i) - (D_i - \hat{m}_0(X_i))\theta), \quad \eta = (g, m)$$

  which satisfies:
  $$\frac{\partial}{\partial \eta} \mathbb{E}[\psi(W_i; \theta, \eta)]\bigg|_{\eta = \eta_0} = 0.$$

  This property ensures that small estimation errors in the nuisance functions $\hat{g}_0(X)$ and $\hat{m}_0(X)$ do not affect the first-order behavior of $\hat{\theta}_0$.

- *Cross-fitting.* To mitigate overfitting, DML uses sample splitting: it estimates $\hat{g}_0(X)$ and $\hat{m}_0(X)$ on one subset of the data, and then uses those estimates to compute $\hat{\theta}_0$ on a disjoint subset. This ensures that residuals are independent of the nuisance function estimates, reinforcing orthogonality and stability.

We will now move to the details of these two sources of bias.

13.2.4 Regularization Bias

For the above PLM specification, a natural idea is to estimate the nuisance function $g_0(X)$ using a machine learning method, and then regress the residual outcome $Y - \hat{g}_0(X)$ on the treatment D. This yields the plug-in estimator:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i \in I} D_i^2 \right)^{-1} \left(\frac{1}{n} \sum_{i \in I} D_i (Y_i - \hat{g}_0(X_i)) \right), \tag{30}$$

where I indexes the main estimation fold in a sample-splitting scheme.

However, as detailed by Chernozhukov et al. (2024a), this naive estimator can suffer from **regularization bias**, which arises because the estimated nuisance function $\hat{g}_0(X)$ typically converges to $g_0(X)$ only at a slow rate, say $n^{-\varphi_g}$ with $\varphi_g < 1/2$.

This slow convergence translates into non-negligible bias in the estimation of θ_0:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) = b + o_P(1),$$

where the bias term b is given by:

$$b = \left(\mathbb{E}[D^2]\right)^{-1} \frac{1}{\sqrt{n}} \sum_{i \in I} m_0(X_i) \left(g_0(X_i) - \hat{g}_0(X_i)\right).$$

This expression shows that unless $\hat{g}_0(X)$ converges faster than $n^{-1/4}$, the bias will dominate the $\sqrt{n}$ scaling and render the estimator inconsistent for inference.

Orthogonality as the Solution. To overcome this challenge, we exploit the concept of **Neyman orthogonality**: by constructing moment equations that are locally insensitive to small errors in the nuisance functions, we eliminate the first-order bias in $\hat{\theta}_0$ due to regularization.

Formally, an orthogonal score satisfies:

$$\frac{\partial}{\partial \eta} \mathbb{E}[\psi(W; \theta, \eta)] \bigg|_{\eta = \eta_0} = 0,$$

ensuring that even if the nuisance estimates $\hat{g}_0, \hat{m}_0$ are imperfect, as long as they converge at a rate faster than $n^{-1/4}$, inference on θ_0 remains valid.

Frisch–Waugh–Lovell (FWL) Theorem. This idea is not new—it was already embedded in the residualization structure of Robinson (1988)'s partially linear estimator and formalized as a geometric result in the **Frisch–Waugh–Lovell (FWL) Theorem**. The FWL theorem states that in a linear model:

$$Y = \beta_0 + \beta_1 D + \beta_2 X + U,$$

the coefficient β_1 can be obtained by:

1. Regressing D on X and taking residuals $\tilde{D} = D - \hat{D}$;
2. Regressing Y on X and taking residuals $\tilde{Y} = Y - \hat{Y}$;
3. Regressing $\tilde{Y}$ on $\tilde{D}$ via OLS.

This "residual-on-residual" regression is equivalent to partialling out X before estimating β_1.

Robinson (1988)'s Innovation on FWL. Replace steps 1-2 with some *non-parametric* regression. Specifically, he does kernel regression in steps 1-2, then do linear regression over step 3.

Chernozhukov et al. (2018)'s Similar Innovation. The goal is to partial out the effect of X from both Y and D, yielding an orthogonalized estimating equation for θ_0:

- Predict D using any $n^{1/4}$-consistent ML method to obtain $\hat{m}_0(X)$.
- Predict Y using another ML method to obtain $\hat{g}_0(X)$.
- Form residuals: $\tilde{V}_i = D_i - \hat{m}_0(X_i)$ and $\tilde{U}_i = Y_i - \hat{g}_0(X_i)$.
- Estimate θ_0 by regressing $\tilde{U}_i$ on $\tilde{V}_i$:

$$\hat{\theta}_0 = \left(\frac{1}{n} \sum_{i=1}^{n} \tilde{V}_i^2 \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} \tilde{V}_i \tilde{U}_i \right). \tag{31}$$

As shown in Chernozhukov et al. (2018), the $\sqrt{n}$-scaled estimation error admits the decomposition:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) = a^* + b^* + c^*,$$

where:

$$\begin{aligned}
a^* &= \left(\mathbb{E}[V^2]\right)^{-1} \frac{1}{\sqrt{n}} \sum_{i=1}^{n} V_i U_i, \quad \text{(mean-zero, asymptotically normal)},\\
b^* &= \left(\mathbb{E}[V^2]\right)^{-1} \frac{1}{\sqrt{n}} \sum_{i=1}^{n} (\hat{m}_0(X_i) - m_0(X_i))(\hat{g}_0(X_i) - g_0(X_i)),\\
c^* &\to 0 \quad \text{if we use cross-fitting.}
\end{aligned}$$

The bias term b^* is second-order if both ML estimators converge at rate $n^{-1/4}$ or faster. The c^* term vanishes when the sample used to estimate the nuisance components is disjoint from the sample used to estimate θ_0—achieved through **cross-fitting**.

Simulation Evidence. Figure 13.2.2 from Chernozhukov et al. (2018) compares the empirical distribution of $\hat{\theta}_0$ from DML versus the naive plug-in approach:

- The non-orthogonal estimator exhibits bias and poor approximation to normality.
- The orthogonalized DML estimator is centered and asymptotically normal.

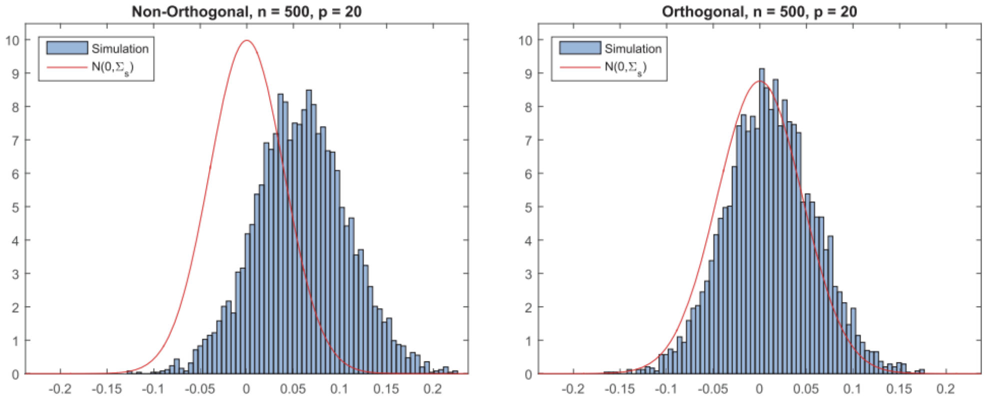


Figure 13.2.2. DML vs Naive Plug-in: Histograms of Estimators in Simulation (from Chernozhukov et al. (2018))

Orthogonalization as Instrumental Variable (IV). Another interpretation of orthogonalization is to view it as constructing an **instrumental variable** for treatment D (Chernozhukov et al., 2018). Specifically, the residualized regressor $\hat{V}_i = D_i - \hat{m}_0(X_i)$ is orthogonal to the error in the regression of Y on $g_0(X)$ but still correlated with D_i.

Thus, the DML estimator:

$$\hat{\theta}_0 = \left(\frac{1}{n}\sum_{i=1}^{n} \hat{V}_i D_i\right)^{-1} \left(\frac{1}{n}\sum_{i=1}^{n} \hat{V}_i (Y_i - \hat{g}_0(X_i))\right)$$

can be interpreted in the spirit of a two-stage least squares estimator, analogous to the classical IV estimator:

$$\hat{\beta}_{\text{IV}} = (Z^\top D)^{-1} Z^\top Y,$$

with $Z_i = \hat{V}_i$ playing the role of the instrument.

13.2.5 Overfitting Bias

To further ensure statistical validity in high-dimensional or flexible ML settings, DML applies **cross-fitting** — a special form of *sample splitting*.

Let I be the index set used to estimate $\hat{g}_0, \hat{m}_0$ and I^c be the sample used to estimate θ_0. The DML estimator becomes:

$$\hat{\theta}_0 = \left(\frac{1}{n}\sum_{i \in I^c} \hat{V}_i D_i\right)^{-1} \left(\frac{1}{n}\sum_{i \in I^c} \hat{V}_i (Y_i - \hat{g}_0(X_i))\right).$$

This construction ensures that the nuisance estimates $\hat{g}_0(X)$ and $\hat{m}_0(X)$ are independent of the sample used to estimate θ_0, breaking the dependency between the regression targets and the estimated nuisance functions. The bias decomposition becomes:

$$\sqrt{n}(\hat{\theta}_0 - \theta_0) = a^* + b^* + c^*,$$

where:

- a^* is asymptotically normal.
- b^* vanishes under sufficient convergence rate of ML estimators.
- c^*, the overfitting bias, vanishes when using cross-fitting.

Formally, the problematic term is:

$$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} V_i(\hat{g}_0(X_i) - g_0(X_i)),$$

which does not converge to zero if the same data is used for both training and estimation. However, under sample splitting:

$$\frac{1}{\sqrt{n}} \sum_{i \in I^c} V_i(\hat{g}_0(X_i) - g_0(X_i)) \xrightarrow{p} 0,$$

because V_i is independent of $\hat{g}_0(X_i)$ by construction, and $\frac{1}{n}\sum_i(\hat{g}_0(X_i) - g_0(X_i))^2 \to 0$ under standard ML consistency conditions.

Figure 13.2.3 from Chernozhukov et al. (2018) contrasts full-sample and split-sample estimation:

- The full-sample approach exhibits severe bias and deviation from asymptotic normality due to overfitting.
- The split-sample estimator is centered and closely follows the theoretical normal distribution.

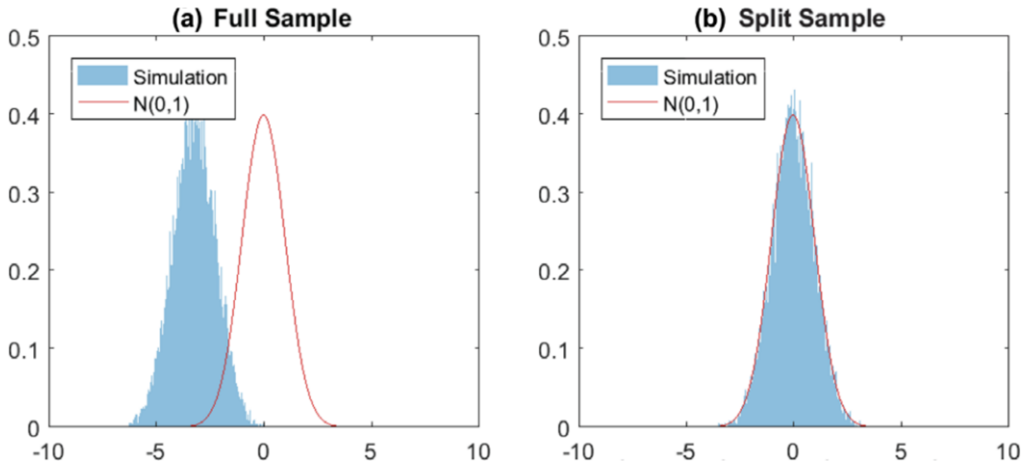


Figure 13.2.3. Comparison of DML Estimators: Full Sample vs Split Sample (from Chernozhukov et al. (2018))

Cross-Fitting to Improve Sample Efficiency. While sample splitting mitigates overfitting bias, it also reduces the effective sample size for estimating treatment effects. To overcome this limitation, the DML framework adopts **cross-fitting**, which generalizes sample splitting in a data-efficient way (Chernozhukov et al., 2018))

Cross-fitting proceeds as follows:

1. Randomly partition the dataset into two subsets.
2. Train ML models $\hat{g}_{0,1}$ and $\hat{m}_{0,1}$ in the first subset.
3. Compute the estimate $\hat{\theta}_{0,1}$ in the second subset using those trained functions.
4. Reverse the roles: train $\hat{g}_{0,2}$ and $\hat{m}_{0,2}$ on the second subset and estimate $\hat{\theta}_{0,2}$ in the first.
5. Average the two estimates: $\hat{\theta}_0 = \frac{1}{2}(\hat{\theta}_{0,1} + \hat{\theta}_{0,2})$.

This procedure reduces bias while restoring estimation efficiency. The method is naturally generalizable to K-fold cross-fitting, allowing the full data to be reused for both training and estimation.

For partial linear models, standard errors can be computed using stratified estimators. For general DML, robust variance estimators are derived in Chernozhukov et al. (2018).

13.2.6 Literature

DML in Action: Practical Guidelines and Empirical Impact. Recent efforts in applied economics and information systems have introduced frameworks for applying

DML in practice. Shi et al. (2024) provide a guide for applied researchers, illustrating the empirical logic, assumptions, and implementation steps in an accessible format.

Their research commentary highlights:

- **Promotion of DML** in business and information systems research.
- **Demonstrations in classical models**: OLS, IV, DiD, and treatment effects models.
- **Clarification of misconceptions**: e.g., incorrect interpretation of machine learning predictions as causal effects.

This "empiricist's guide" clarifies how DML retains robustness under complex covariate structures, especially when traditional models would suffer from omitted variable bias or inefficient functional form assumptions.

Case Study: Labor Supply Elasticity with Monopsony. A powerful empirical illustration of DML is given by Dubé et al. (2023), who estimate labor supply elasticity in online gig markets.

In their partial linear model setup, wage (the treatment D) affects labor supply (the outcome Y), controlling for high-dimensional covariates Z (e.g., geography, worker attributes). They adopt the standard DML estimator:

$$(1) \quad \texttt{log(earnings)} = m_0(Z) + \theta \cdot \texttt{log(wage)} + \varepsilon, \quad \mathbb{E}[\varepsilon|Z] = 0, \tag{32}$$

$$(2) \quad \hat{\theta} = \left(\frac{1}{n}\sum \hat{V}_i^2\right)^{-1} \cdot \left(\frac{1}{n}\sum \hat{V}_i\hat{U}_i\right), \tag{33}$$

where $\hat{V}_i$ and $\hat{U}_i$ are residuals from regressing D and Y on Z.

Their results, supported by follow-up experiments, validate the DML framework's robustness in identifying elasticities that are difficult to estimate in standard regression settings.

Case Study: Mobile Payment Adoption and Credit Card Usage. Xu et al. (2023) investigate the causal impact of adopting Alipay, a prominent mobile payment platform, on consumers' credit card usage behavior. Utilizing a unique dataset from a leading Asian bank, they analyze credit card transactions before and after customers adopt Alipay's mobile payment services.

Specifically, their study implements DML in the context of a partially linear model in the aforementioned specification, where

- Y_i denotes the outcome variable (e.g., credit card transaction amount),
- D_i represents the treatment indicator (Alipay adoption),

- X_i is a vector of high-dimensional covariates,
- $g_0(\cdot)$ and $m_0(\cdot)$ are unknown nuisance functions,
- ε_i and ν_i are error terms.

The DML procedure involves the following steps:

1. **Nuisance Estimation:** Employ machine learning algorithms (e.g., random forests, boosting) to estimate the nuisance functions $\hat{g}_0(X_i)$ and $\hat{m}_0(X_i)$.
2. **Orthogonalization:** Compute residuals;
3. **Estimation:** Regress $\tilde{Y}_i$ on $\tilde{D}_i$ to obtain an estimate of the treatment effect θ_0.

This orthogonalization ensures that the estimation of θ_0 is robust to errors in the nuisance function estimations, provided these errors converge sufficiently fast.

They find that

- **Increased Credit Card Activity:** Post-adoption, customers exhibit a 9.4% increase in total credit card transaction amounts and a 10.7% increase in transaction frequency at the focal bank.
- **Enhanced Customer Loyalty:** The adoption of Alipay reduces customer churn, indicating improved loyalty to the bank.
- **Channel Substitution and Complementarity:** Alipay serves as a substitute for physical card payments in offline channels and complements PC-based online payments, suggesting a shift in consumer payment behavior.

Case Study: Reputation and Persuasion in Online Debates. Manzoor et al. (2023) examine the causal impact of *reputation* on the success of persuasion in large-scale online deliberation. Using a seven-year panel of over one million debates from a structured argumentation platform, the authors focus on isolating the *ethos effect* — the persuasive power of a speaker's reputation, independent of the content and quality of the argument.

To credibly identify the causal effect of reputation, the study confronts two empirical challenges:

1. Unobserved confounding(A) due to latent speaker characteristics (e.g., inherent eloquence or social status).
2. High-dimensional control variables(X) embedded in *argument text*, which are semantically rich but unstructured.

The authors resolve these using a two-stage estimation strategy grounded in the **Double Machine Learning** (DML) framework:

- They construct an **instrument**(Z) for reputation based on exogenous variation from past debate competition dynamics.
- They model the unstructured argument content using pretrained **transformer-based neural language models**, extracting contextualized embeddings.
- They apply DML to estimate the treatment effect of reputation on persuasion, orthogonalizing out high-dimensional linguistic controls.

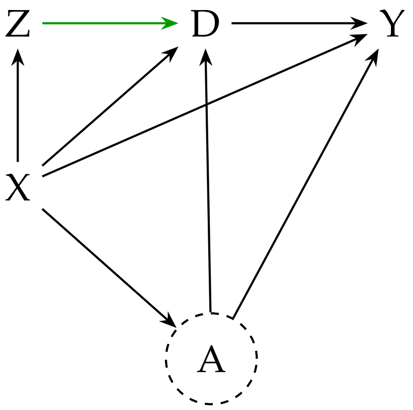


Figure 13.2.4. Causal Graph for Manzoor et al. (2023), with IV

Case Study: Advertising Measurement at Scale on Facebook. Gordon et al. (2023) explore whether double machine learning (DML) and *stratified propensity score matching* (SPSM, see last section) can recover randomized controlled trial (RCT) estimates of ad effectiveness in large-scale observational data collected on Facebook.

The authors leverage a dataset of 663 large-scale experiments conducted by Facebook, featuring over 5,000 user-level features. They benchmark DML and SPSM against ground-truth experimental results across advertising funnel stages.

Their results show significant limitations:

- Median RCT lift is 20%, 18%, and 5% across upper, mid, and lower funnel outcomes.
- However, using DML (with SPSM), the median bias in recovered estimates is 83%, 58%, and 24%, respectively—indicating substantial deviations from experimental benchmarks.

They conclude that even SOTA learning methods like DML cannot reliably estimate causal effects in this context due to multiple failures in underlying assumptions (e.g., unconfoundedness, overlap). The implication is that observational methods, even when enhanced with modern causal ML frameworks, struggle to replicate experimental results in digital advertising settings.[10]

This study exemplifies the limits of DML in high-stakes applied field contexts and emphasizes the need for rigorous experimental validation.

Reliability of DML Estimators. Recent advances in **DML** have provided researchers with powerful tools to perform valid causal inference in high-dimensional and complex observational settings. However, as field applications grow, the credibility and robustness of these methods require careful empirical validation.

Fuhr et al. (2024) offer such an evaluation by comparing DML with a suite of traditional and modern estimators across simulations and real-world settings, including estimating the effect of air pollution on housing prices. Their findings reveal two key insights:

- DML estimates are **consistently larger** than those from less flexible methods like OLS, reflecting its capacity to adjust for complex, nonlinear confounding.
- Despite its flexibility, the performance of DML still hinges on standard identification assumptions (e.g., unconfoundedness), and it is sensitive to choices like cross-fitting, model tuning, and sample size.

Overall, while DML holds promise for empirical research in economics, marketing, and information systems, its application must be accompanied by transparency, careful diagnostics, and robustness checks. The method's strength lies in its balance between modern ML techniques and rigorous causal identification strategies.

13.3 Generic Framework of DML

In the preceding sections, we introduced the concept of **Neyman orthogonality**, highlighting its role in addressing regularization bias by constructing score functions that are robust to small estimation errors in high-dimensional nuisance components. This sets the stage for moving beyond the illustrative Partial Linear Model (PLM) and toward a more general and unified framework for **Double Machine Learning (DML)**.

While the PLM provides a valuable baseline for understanding DML's mechanics, it imposes a separable structure between the treatment D and covariates X, and assumes linear effects for the target parameter θ_0. Many practical applications involve:

- More complex causal relationships not captured by linear models,

[10] Echoing the in-class question:Which of the assumptions for DML are violated in the FB observational data?

- Flexible treatment or instrument functions,
- High-dimensional or non-tabular data structures (e.g., text, images).

Therefore, in this part, we introduce a more general DML framework, retaining the same three methodological pillars:

1. **Score Function Construction:** Define an orthogonal score $\psi(W;\theta,\eta)$ that satisfies the Neyman orthogonality condition:
$$\partial_\eta \mathbb{E}[\psi(W;\theta,\eta)]\big|_{\eta=\eta_0} = 0.$$
2. **Sample Splitting and Cross-Fitting:** Use disjoint subsamples for nuisance parameter estimation and target parameter estimation to remove overfitting bias.
3. **Machine Learning for Nuisance Estimation:** Allow $\hat{\eta}_0$ (e.g., outcome regression $\mu_w(X)$ and propensity score $e(X)$) to be estimated using arbitrary ML methods, subject to mild convergence rate assumptions.

13.3.1 Revisiting Neyman Orthogonality

As a general framework, we consider DML estimation and inference for some low-dimensional target parameter θ_0 based upon the empirical analog of the moment condition, or its score function:

$$\mathbb{E}[\psi(W;\theta_0;\eta_0)] \tag{34}$$

We resume from the questions:

- Why does Neyman orthogonality solve the regularization bias?

  - Neyman orthogonality solves regularization bias by ensuring that small errors in estimating nuisance functions (e.g., outcome models or propensity scores) have only a second-order effect[11] on the estimation of the target parameter θ_0. Such that, it makes the moment condition (or score function) locally insensitive to estimation error in the nuisance parameters η. So even if $\hat{\eta}$ is biased due to regularization (as in Lasso or random forests), the bias does not propagate to $\hat{\theta}_0$ at first order—preserving consistency and root-n inference.

  If the score function used to estimate θ_0 satisfies **Neyman orthogonality**—that is, the first-order derivative of the expected score function with respect to the

[11] A second-order effect means the impact of an error is quadratically small—that is, if the error in estimating the nuisance function $\hat{\eta}$ is of order ε, then the resulting error in the target parameter $\hat{\theta}$ is of order ε^2.

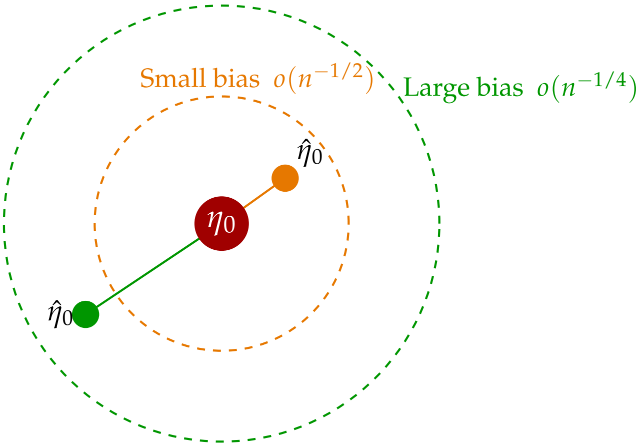


Figure 13.3.1. Neyman Orthogonality on Regularization Bias

nuisance parameter is zero at the true value—then the influence of nuisance estimation error enters only as a *second-order effect*.

This means even when $\hat{\eta}_0$ is not very close to η_0 (i.e., in the "large bias" green region with convergence rate $o(n^{-1/4})$), the bias in $\hat{\theta}_0$ remains small enough that $\hat{\theta}_0$ can still be **root-n consistent** and asymptotically normal.

In summary, Neyman orthogonality decouples the estimation of the treatment effect from small errors in nuisance parameters. It makes the bias from estimating $\hat{\eta}$ a second-order effect, which decays fast enough to preserve root-n consistency.[12]

- How it could work to eliminate the overfitting bias?

  Overfitting occurs when the same data is used to both train and evaluate the estimator, leading to biased score functions and invalid inference. We have the following procedure:

  1. Randomly partition the sample into K equal folds $\{I_k\}_{k=1}^K$ of size $n = N/K$.
  2. For each fold k, estimate nuisance parameters $\hat{\eta}_{0,k}$ using only the complement I_k^c.
  3. Evaluate the score function on I_k using $\hat{\eta}_{0,k}$, then aggregate:

  $$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{n,k}\left[\psi(W;\tilde{\theta}_0,\hat{\eta}_{0,k})\right] = 0,$$

     and solve for the final estimator $\tilde{\theta}_0$.

  This ensures that:

[12] Full mathy version seen in appendices.

- The nuisance parameter estimates $\hat{\eta}_{0,k}$ are *not trained on the evaluation sample*.
- This separation reduces dependency between estimation errors and score evaluation.
- It prevents overfitting bias, preserves Neyman orthogonality, and ensures root-n consistency of $\tilde{\theta}_0$ under weak conditions.

13.3.2 Beyond PLM: Double Machine Learning in Interactive Regression Models

Having introduced the core idea of **Neyman orthogonality**, which ensures that the estimation of nuisance parameters does not contaminate the target parameter estimate at the root-n rate, we now extend our scope to more general frameworks. One such important generalization is the *Interactive Regression Model* (IRM), which is capable of capturing heterogeneous treatment effects.

Setup. In the IRM framework, the observed outcome Y depends on both treatment $D \in \{0,1\}$ and covariates X, through a potentially nonlinear interaction function $g_0(D,X)$. The treatment itself may depend on covariates through a propensity score $m_0(X)$. Formally, the model takes the form:

$$\begin{aligned} Y &= g_0(D,X) + \varepsilon, \quad \mathbb{E}[\varepsilon \mid D,X] = 0, \\ D &= m_0(X) + \tilde{D}, \quad \mathbb{E}[\tilde{D} \mid X] = 0. \end{aligned}$$

Target parameter. The parameter of interest is the *average predictive effect (APE)*:

$$\theta_0 = \mathbb{E}[g_0(1,X) - g_0(0,X)],$$

which coincides with the average treatment effect (ATE) under conditional exogeneity.

Estimation challenge. Since g_0 and m_0 are unknown and can be high-dimensional or nonparametric, machine learning tools are naturally suited for estimating them. However, this introduces **regularization bias** and the risk of **overfitting**.

DML Solution. To tackle these issues, the DML approach introduces:

- **Orthogonal score functions:** Defined to be insensitive to small errors in the nuisance function estimates $\hat{g}, \hat{m}$, ensuring second-order bias control.
- **Sample splitting and cross-fitting:** Training nuisance functions on one fold and evaluating the score on another to eliminate overfitting bias.

Score function. A typical orthogonal moment function used in IRM-based DML for ATE is:

$$\psi(W;\theta,\eta) := (g(1,X) - g(0,X)) + \frac{D(Y-g(1,X))}{m(X)} - \frac{(1-D)(Y-g(0,X))}{1-m(X)} - \theta$$

where $\eta = (g,m)$, $\eta_0 = (g_0, m_0)$.

Key step. Thanks to the Neyman orthogonality condition

$$\partial_\eta \mathbb{E}[\psi(W;\theta_0,\eta)]\big|_{\eta=\eta_0} = 0,$$

the DML estimator of θ_0 remains root-n consistent even if the nuisance components $\hat{g}, \hat{m}$ converge only at slower rates (e.g., $n^{-1/4}$ in L_2 norm).

Reference. For a detailed theoretical treatment and empirical demonstration of this framework, see Chapter 9.3 of Chernozhukov et al. (2024b), where the IRM-DML method is formally introduced and analyzed.

13.3.3 Bias, Variances via Neyman Orthogonality

A central insight of the Double Machine Learning (DML) framework is that causal estimators constructed from **Neyman orthogonal score functions** are naturally equipped with bias correction mechanisms. This idea has been formalized in both classical semiparametric inference and its modern ML-powered adaptations.

The key is the decomposition of the estimator into two parts:

- A naive plug-in term using estimated nuisance functions, which may be biased,
- A correction term, typically involving inverse-propensity weighting (IPW), which offsets the bias.

Formally, for estimating the Average Treatment Effect (ATE), the Augmented Inverse Propensity Weighting (AIPW) estimator is expressed as:

$$\hat{\theta}_{\text{AIPW}} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{W_i(Y_i - \hat{\mu}_1(X_i))}{\hat{e}(X_i)} - \frac{(1-W_i)(Y_i - \hat{\mu}_0(X_i))}{1-\hat{e}(X_i)} + \hat{\mu}_1(X_i) - \hat{\mu}_0(X_i)\right],$$

where $\hat{e}(X_i)$ is the estimated propensity score, and $\hat{\mu}_w(X_i)$ are estimated outcome regressions.

The last two terms constitute a *bias-correction* to the plug-in regression adjustment. When the score function is orthogonal (in the Neyman sense), the impact of estimation errors in the nuisance functions on the treatment effect estimator is minimized. This is critical because ML-based nuisance function estimators are often biased due to regularization and overfitting.

Variance and Statistical Properties The asymptotic variance of the DML estimator $\hat{\theta}_{\text{DML}}$ is given by the empirical variance of the orthogonal score:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} \hat{\psi}_i^2,$$

where $\hat{\psi}_i$ is the influence function derived from the Neyman-orthogonal moment condition.

Thanks to the orthogonality, this estimator is:

- **Root-n consistent**,
- **Asymptotically normal**, under mild regularity conditions,
- **Robust** to slow convergence of nuisance estimates, as long as they satisfy the $o(n^{-1/4})$ convergence rate.

Connection to AIPW and General Frameworks The AIPW estimator can be interpreted as a special case of the DML estimator using plug-in nuisance function estimates combined with a bias-correction term from the IPW structure. This aligns with results from Chernozhukov et al. (2018) and Belloni et al. (2016), who prove that DML estimators under orthogonality and cross-fitting deliver valid inference even with complex, high-dimensional X.

DML with Deep Learning Recent work extends this framework by integrating deep neural networks (DNNs) for flexible estimation of nuisance components. When DNNs achieve a convergence rate of $o(n^{-1/4})$ under appropriate smoothness assumptions (e.g., β-smoothness of the target functions), DML retains its root-n consistency (Farrell et al., 2021).

This bias correction logic remains intact even in these nonparametric, high-dimensional contexts, thus validating the wide applicability of DML in modern empirical research.

13.3.4 Literature

DNN and Deep Learning. Farrell et al. (2021) show that under mild smoothness assumptions, the empirical risk minimizer for the DNN class converges uniformly with high probability:

Theorem 13.1 (Multilayer Perceptron Convergence). *Suppose the empirical minimizer exists, define $\widehat{f}_{MLP} \in \arg\min_{f \in \mathcal{F}_{MLP}} \sum_{i=1}^{n} \ell(f(z_i))$, where $\mathcal{F}_{MLP}$ denotes the class of MLP functions with ReLU activations, fixed architecture depth L, and width $H \asymp n^{\frac{1}{2\beta+d}} \log^2 n$.*

And then, under Assumptions 1 and 2 in Farrell et al. (2021), for large n, with high probability:

$$\|\widehat{f}_{MLP} - f_0\|_{L_2(X)}^2 \leq C \cdot \left(n^{-\frac{2\beta}{2\beta+d}} \log^2 n + \frac{\log n}{n} \right),$$

where β is the smoothness of f_0 and C is a constant.

Remark. *The total number of parameters in such networks is $W = (d+1)H + (L-1)(H^2 + H) + H + 1$, which can grow astronomically large.*

For causal inference, the paper proposes a **Neyman orthogonal score function** for the average potential outcome:

$$\psi_t(z) = \frac{\mathbb{1}\{t_i = t\}}{\widehat{p}_t(x_i)} (y_i - \widehat{\mu}_t(x_i)) + \widehat{\mu}_t(x_i), \tag{35}$$

$$\widehat{\tau} = \mathbb{E}_n \left[\widehat{\psi}_1(z_i) - \widehat{\psi}_0(z_i) \right], \tag{36}$$

where $\widehat{p}_t(x)$ is the estimated propensity score, and $\widehat{\mu}_t(x)$ the estimated outcome regression (both learned via DNNs).

They prove that this estimator is root-n consistent and asymptotically normal:

$$\sqrt{n} \, (\widehat{\tau} - \tau) \xrightarrow{d} \mathcal{N}(0, \Sigma),$$

where the variance Σ involves both outcome and propensity score estimation errors.

Semi-parametric DML with Deep Learning for Heterogeneous Effects Farrell et al. (2020) propose a general inference framework that integrates deep learning models into semiparametric causal estimation. Their goal is to estimate heterogeneous treatment effects $\theta_0(X)$ while accounting for rich, high-dimensional covariates X and maintaining valid statistical inference.

Their setting is based on a semiparametric data-generating process (DGP) where treatment assignment may follow complex patterns, but the structural model (e.g., the outcome or treatment response function) is parametrically structured.

The key insight is to treat the structural parameter $\theta_0(X)$ as a function-valued object to be estimated via deep learning. The neural network is trained not to predict outcomes directly but to minimize an orthogonalized loss associated with a moment function that satisfies Neyman orthogonality. They organize the architecture as:

- Inputs: $x_1, \ldots, x_d$ representing high-dimensional covariates.

- Hidden layers: Standard ReLU layers.
- Parameter layer: Encodes $\hat{\theta}_t(X)$ for each treatment arm t.
- Model layer: Computes outcome y from treatment t and covariates.

To ensure robust inference, they construct an orthogonal score:

$$\psi(w, \theta, \Lambda) = H(x, \theta(x); t^*) - H_0(x, \theta(x); t^*)\Lambda(x)^{-1}\ell_\theta(y, t, \theta(x)),$$

where

$$\mu_0 = \mathbb{E}[H(X, \theta_0(X); t^*)]$$

and ℓ_θ is a smooth per-observation loss.

This score is Neyman orthogonal with respect to nuisance components $\Lambda(x)$, meaning small estimation errors in Λ do not affect the first-order behavior of the estimator.

- The framework automates valid inference for heterogeneous treatment effects with DNNs.
- Achieves $\sqrt{n}$-consistency and asymptotic normality under minimal smoothness assumptions.
- Integrates score-based methods (influence functions) with modern deep learning optimizers and architectures.

Unlike standard prediction tasks, their implementation separates the parameter layer from the loss—this allows $\theta_0(X)$ to be trained via orthogonalized scores rather than direct supervised loss minimization.

An Empiricist's Guide to DML A recent advancement in the application of Double Machine Learning (DML) is presented in the work of Ye et al. (2023), who propose the DeDL (Debiased Deep Learning) framework to operationalize DML in large-scale field experiments. Their setting addresses a practically important question: how can we estimate the causal effect of any treatment combination when only a subset of all possible combinations is observed?

The DeDL framework follows a four-step procedure grounded in the DML tradition but tailored for empirical scalability and treatment heterogeneity:

- Step 1: Specify the DGP. The data-generating process is modeled as

$$\mathbb{E}[Y \mid X = x, T = t] = G(\theta^*(x), t),$$

where $G(\cdot,\cdot)$ is a known structural function and $\theta^*(x)$ is a high-dimensional, individual-level parameter to be estimated. This specification allows the model to accommodate rich heterogeneity while preserving semiparametric structure.

- Step 2: Train the DNN. The deep neural network is trained to estimate $\theta^*(x)$ by minimizing the squared loss:

  $$\hat{\theta}(x) := \arg\min_{\theta \in \mathcal{F}_{\text{DNN}}} \frac{1}{n} \sum_{i=1}^{n} (y_i - G(\theta(x_i), t_i))^2 .$$

  This approach allows for flexible treatment assignment while enforcing parametric structure on outcome modeling.

- Step 3: Derive Neyman-Orthogonal Score Functions. Using the estimated $\hat{\theta}(x)$, the authors construct influence functions to identify average treatment effects (ATE) and optimal treatment rules. Let t^* be the best treatment according to the model, and define:

  $$\psi(z,\theta,\Lambda) := H(z,\theta(x),t) - H_0(z,\theta(x);t^*)\Lambda(x)^{-1}\ell_\theta(y,t,\theta(x)),$$

  where H is the conditional outcome model, and ℓ is the loss function. This setup ensures orthogonality with respect to first-step estimation errors.

- Step 4: Estimation and Inference. Cross-fitting is applied to obtain a final root-n consistent estimator:

  $$\sqrt{n}(\hat{\theta}_{\text{DeDL}}(t) - \theta(t)) \xrightarrow{d} \mathcal{N}(0,1).$$

  The DeDL procedure enables valid inference on $\theta(t)$ despite high-dimensional nuisance estimation.

To address empirical deviations from random assignment, the authors propose stratified sampling: the covariate space is partitioned into over 69,000 strata, with users sampled uniformly across strata within each treatment group. This effectively reweights the empirical design to resemble the intended randomized experiment.

Finally, DNN cross-validation error is used to monitor convergence and tuning. In practice, the DeDL estimator significantly outperforms standard DML and semi-debiased alternatives on measures of MSE and mean absolute percentage error (MAPE), as demonstrated in their real-world experiments.

This framework not only operationalizes theoretical guarantees of DML but also provides a robust empirical playbook for practitioners implementing causal inference with deep learning.

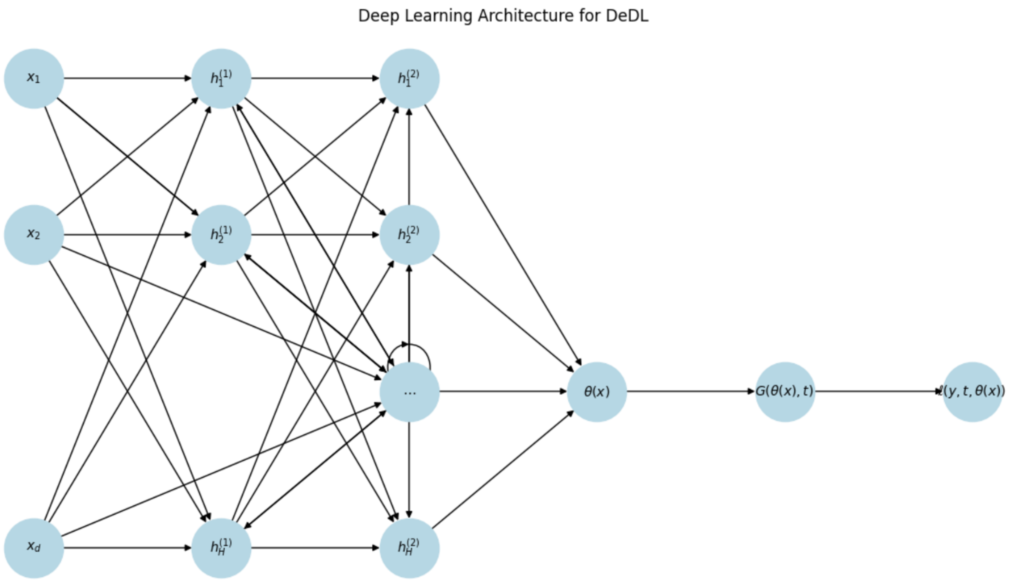


Figure 13.3.2. Deep Learning Architecture For DeDL

13.4 DML: Good News and Caveats

Good News. Double Machine Learning (DML) has shown particularly strong empirical performance in complex settings where:

- The data generating process (DGP) $G(\cdot)$ is high-dimensional or nonlinear.
- There are many possible treatment combinations, as in large-scale A/B testing.
- Even with a nontrivial DNN error $\mathbb{E}_X \|\hat{\theta}(X) - \theta^*(X)\|_2$, **Debiased DNN** methods like DeDL yield valid inference, provided the DGP $G(\cdot)$ is correctly specified.

Bad News.

- When the link function $G(\cdot)$ is mis-specified, DeDL can **exacerbate** rather than correct the errors of the underlying DNN, leading to poor inference performance.
- Practitioners must resort to trial-and-error or cross-validation to identify a valid DGP structure.

Recent developments generalize DeDL to nonparametric nuisance specifications, allowing automated debiasing even under model ambiguity (Chernozhukov et al., 2022). The rough idea of this work is as follows, suppose the target parameter is $\theta_0 = \mathbb{E}[m(W, \gamma)]$, and . It represents the parameter of interest into one Riesz representer $\alpha_0(X)$ andthe conditional mean $\gamma_0(x) = \mathbb{E}[Y|X = x]$ as follows,

$$\mathbb{E}[\alpha_0(X)Y(X)] = \theta_0 \quad \text{for all } Y \text{ such that } \mathbb{E}|Y(X)|^2 < \infty,$$

where $\alpha_0(X)$ can be estimated via base functions: $\hat{\alpha}(x) = \frac{1}{n-n_t}\sum_{i\in I_t}[m(W_i,1)+b(x)\hat{\rho}_t]$, where $\hat{\rho}_t = \arg\min\{-2M_t\rho + \rho'\hat{G}_t\rho + 2\sum_{j=1}^{L}|\rho_j|\}$, $M_t = \frac{1}{n-n_t}\sum_{i\in I_t} m(W_i,b_i)$, $\hat{G}_t = \frac{1}{n-n_t}\sum_{i\in I_t} b(x_i)b(x_i)'$.

With this representation, the **Neyman orthogonal score function** becomes:

$$\psi(w,\theta,\gamma,\alpha) = m(w,\gamma) - \theta + \alpha(x)[y-\gamma(x)].$$

This score delivers robustness even under complex nuisance estimation and helps facilitate bias correction and valid standard error computation.

13.4.1 DML for Difference-in-Differences (DiD)

Recent developments apply DML to DiD frameworks(Chang, 2020), offering valid inference even with high-dimensional covariates and flexible nuisance components.

Model. The data-generating process takes the form:

$$Y_{it} = \mu + X_i^{\top}\zeta + \tau\cdot D_{it} + \delta_t + \alpha_i + \varepsilon_{it}, \quad D_{it} = \mathbb{1}\{T_i = t\}$$

The **Average Treatment Effect on the Treated (ATT)** is:

$$\theta_{\text{ATT}} = \mathbb{E}[Y_{it}(1) - Y_{it}(0) \mid D_{it} = 1]$$

The new score function for repeated cross sections data structure[13] is defined as:

$$\psi(W,\theta_0,p_0,\lambda_0,\eta_0) = \frac{T-\lambda_0}{\lambda_0(1-\lambda_0)}\cdot\frac{Y}{P(D=1\mid X)}\cdot\frac{D-P(D=1\mid X)}{1-P(D=1\mid X)} - \theta_0 - c,$$

where the adjustment term c is:

$$c = \frac{D-P(D=1\mid X)}{\lambda_0(1-\lambda_0)\cdot P(D=1\mid X)\cdot(1-P(D=1\mid X))}\times E[(T-\lambda_0)Y\mid X, D=0].$$

and the nuisance parameters include the unknown constants $p_0 = P(D=1)$ and $\lambda_0 = P(T=1)$, and the unknown function:

$$\eta_0 = (P(D=1\mid X), E[(T-\lambda)Y\mid X, D=0]) = (g_0,\ell_0).$$

[13] Repeated cross-sectional data consists of multiple independent cross-sections collected at different points in time. Unlike panel data, where the same individuals are tracked over time, repeated cross-sections draw a fresh sample in each wave, i.e., Educational Surveys which sample different groups of students each year to assess learning outcomes. This approach allows researchers to analyze aggregate trends over time, but it does not track individual-level changes. For further explanation, please check the great markdown link.

This flexible DML-DiD framework enables:

- Treatment effect estimation under panel and repeated cross-section designs.
- Robustness to flexible covariate adjustment.
- Plug-in and orthogonalized score functions for valid inference.

13.5 The Notebooks

- The notebook FWL_Theorem tests the equivalence of FWL-type regression and the OLS regression numerically;
- The notebook DML_EconML_ATE implements double machine learning methods using EconML which is developed by Microsoft.

14 Heterogeneous Treatment Effects (HTE)

The study of heterogeneous treatment effects (HTE) aims to understand how and why causal effects differ across individuals or subgroups in a population. This is in contrast to traditional causal inference, which focuses on estimating average treatment effects (ATE) across an entire sample. In many real-world applications—ranging from personalized medicine to targeted advertising to fairness-aware machine learning—it is critical not only to know whether a treatment works on average, but for whom it works and under what conditions. We now introduce the fundamental motivations behind the study of HTE.

14.1 From Average Treatment Effect to Conditional Effect

14.1.1 The Classical Setup

In the Neyman–Rubin potential outcomes framework, let $Y(1)$ and $Y(0)$ denote the potential outcomes under treatment and control, respectively. For each unit i, we define the individual treatment effect as:

$$\tau_i = Y_i(1) - Y_i(0)$$

However, because we can never observe both $Y_i(1)$ and $Y_i(0)$ for the same unit, individual-level causal effects are unobservable. The usual estimand becomes the average treatment effect (ATE):

$$\text{ATE} = \mathbb{E}[Y(1) - Y(0)]$$

While useful, the ATE provides only an aggregate measure. It conceals important heterogeneity—individuals or subgroups may benefit more or less from the same intervention.

14.1.2 Conditional Average Treatment Effect (CATE)

To address heterogeneity, we instead estimate the Conditional Average Treatment Effect (CATE):

$$\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]$$

Here, X is a vector of observed covariates that capture characteristics of the unit (e.g., age, income, risk aversion, pre-treatment health status). The function $\tau(x)$ is known as the heterogeneous treatment function, and estimating it lies at the heart of HTE.

HTE for Theoretical Insight and Explanation

Moderators and Mechanisms In social science and economics, understanding why a treatment works is often as important as whether it works. Theory frequently suggests that a treatment's effectiveness depends on observable characteristics—so-called moderators.

For example:

- A job training program might only help low-skilled workers.
- A health campaign may be more effective for individuals with higher baseline knowledge.

HTE analysis helps test these kinds of moderation hypotheses. It provides evidence for causal mechanisms and refines policy targeting by identifying which subgroups benefit most.

Empirical Goal In such settings, estimating heterogeneity is not just an auxiliary task—it is the primary object of interest. We are concerned with identifying variables that drive variation in treatment effects and with formally testing interactions between treatment and covariates.

HTE estimation is an empirical route to test structural hypotheses embedded in theoretical models.

HTE for Prescription and Personalization

Resource Allocation Under Constraints In many settings, treatments are costly, risky, or limited in supply. Allocating them efficiently requires knowing not only the average effect but how it varies.

Let $\pi(x) \in \{0,1\}$ be a policy function that assigns treatment based on covariates x. Then the optimal policy (under no cost of treatment) is:

$$\pi^*(x) = \mathbb{I}[\tau(x) > 0]$$

This shows that estimating $\tau(x)$ accurately is crucial for optimal targeting.

Key Domains

- **Personalized medicine:** Should this patient receive immunotherapy based on their genomic profile?
- **Targeted marketing:** Which customer segments should receive a promotion?

- **Education:** Which students benefit most from a remedial program?

In all of these, heterogeneous responses are expected, and modeling them improves outcomes.

HTE in Propensity-Score-Based Estimators

Even classical methods such as:

- Inverse Probability Weighting (IPW)
- Augmented IPW (AIPW)
- Propensity Score Matching (PSM)
- Post-stratification

implicitly condition on observed covariates. While designed for estimating the ATE, they often produce localized estimates of CATE in areas of sufficient overlap in propensity scores.

Therefore, even within conventional frameworks, HTE is already present—just not explicitly framed or targeted.

HTE and Fairness in Machine Learning

In ML-driven decision systems, fairness has become a primary concern. If the effect of an algorithmically assigned treatment (e.g., job offer, loan approval) varies systematically across protected groups (e.g., gender, race), then the system may amplify social inequalities.

HTE provides tools to quantify and audit such issues:

- Are women consistently benefiting less from an online course?
- Do low-income borrowers benefit more from financial literacy interventions?

These questions fall under the broader theme of algorithmic fairness through causal lenses.

14.2 Overview of HTE Estimation Literature

There are three major approaches to estimating heterogeneous treatment effects (HTE), each rooted in a different academic tradition—econometrics, statistics, and computer science. These methods differ in how directly they model heterogeneity, their assumptions, and their suitability for causal inference or prediction.

14.2.1 Causal Trees and Causal Forests

The first category includes tree-based models specifically designed to detect variation in treatment effects across subgroups:

- **Causal Trees**(Athey and Imbens, 2016): modifies traditional decision tree algorithms (like CART) to focus on identifying subgroups with differing treatment effects rather than maximizing predictive accuracy;
- **Causal Forests**(Wager and Athey, 2018), build on causal trees by aggregating many trees (as in Random Forests) to stabilize predictions and allow for valid inference (e.g., confidence intervals for CATEs).

These traditional methods, though be

- Flexible and nonparametric, suitable for high-dimensional covariates.
- Good for discovering interpretable subgroups.
- Valid inference with honest estimation.

but are also

- Typically limited to binary treatments.
- Confidence intervals can be wide in small samples.
- Relies on unconfoundedness and SUTVA assumptions.

We would go over them in the next sections.

14.2.2 Double Machine Learning (DML)

Double Machine Learning (DML)(Chernozhukov et al., 2018) is a framework that combines orthogonalization and sample splitting to debias treatment effect estimation using machine learning models. We have studied it over the last chapter. Originally developed to estimate average treatment effects (ATE), DML can also be used for CATE by leveraging its nuisance parameter models (e.g., propensity scores, outcome regressions) and plugging them into flexible ML-based meta-learners. They are

- Robust to high-dimensional confounders.
- Compatible with many ML algorithms (Lasso, Boosting, Neural Networks).
- Theoretical guarantees (root-n consistency, asymptotic normality).

while

- Primarily designed for ATE.
- Requires careful cross-fitting and implementation.

14.2.3 Uplift Modeling and Meta-learners

This approach, popular in industry, aims to estimate the treatment effect directly as the difference in response probability under treatment and control.

The **two-model** strategy trains separate predictive models for treated and control groups. More principled versions include **meta-learners** such as the T-learner, S-learner, and X-learner(Künzel et al., 2019), which formalize this approach using supervised learning. They are

- Easy to implement using standard ML pipelines.
- Highly scalable for A/B testing and marketing.

while

- Often lack statistical inference (e.g., confidence intervals).
- May not respect causal assumptions (e.g., ignorability).

Table 12. Comparison of HTE Estimation Methods

Method	**Best Use Case**	**Key Limitations**
Causal Trees & Forests	Subgroup discovery with interpretable structure	Binary treatment only; requires honest sample splitting for valid inference
Double Machine Learning	Semiparametric estimation with theoretical robustness and flexible ML tools	Primarily focused on ATE; implementation can be complex
Uplift & Meta-learners	Fast deployment in real-world business contexts such as marketing personalization and A/B testing	Do not yield valid confidence intervals; rely on strong assumptions for causal validity

Each method reflects a different trade-off between flexibility, causal identification, and scalability. Choosing the right approach depends on your goal: are you testing a theory, targeting individuals, or auditing fairness?

14.3 Causal Tree and Causal Forest Methods

Machine learning has introduced powerful nonparametric methods for estimating heterogeneous treatment effects (HTEs). Among them, **Causal Trees** and **Causal Forests** represent a foundational family of algorithms that directly model treatment heterogeneity using recursive partitioning techniques. These models are particularly useful when the analyst aims to uncover interpretable subgroups of the population whose responses to treatment differ significantly.

14.3.1 *The Causal Tree Algorithm*

Proposed by Athey and Imbens (2016), the causal tree framework modifies the classical CART (Classification and Regression Trees) algorithm for causal inference tasks. Instead of minimizing prediction error, the algorithm seeks to partition the covariate space into regions where the treatment effect varies the most. The key idea is to treat the *treatment effect itself* as the quantity to be predicted and optimized.

Formally, given data $\{(X_i, Y_i, D_i)\}_{i=1}^n$, where $X_i \in \mathbb{R}^p$ is a vector of covariates, $D_i \in \{0, 1\}$ is the binary treatment indicator, and $Y_i \in \mathbb{R}$ is the observed outcome, we assume the unconfoundedness condition:

$$(Y_i(0), Y_i(1)) \perp\!\!\!\perp D_i \mid X_i$$

The tree recursively partitions the space of X to create leaves $L \in \mathcal{L}$, and within each leaf, the treatment effect is estimated as:

$$\hat{\tau}_L = \frac{1}{n_{L,1}} \sum_{i \in L, D_i = 1} Y_i - \frac{1}{n_{L,0}} \sum_{i \in L, D_i = 0} Y_i$$

where $n_{L,1}$ and $n_{L,0}$ denote the number of treated and control units in leaf L, respectively.

Splitting Criterion: Instead of using mean squared error or Gini impurity, causal trees split on covariates that maximize heterogeneity in estimated treatment effects between potential child nodes.

Interpretability: The resulting tree provides an interpretable decision rule for identifying subgroups with differing treatment effects.

14.3.2 *Limitations of Causal Trees*

Despite their conceptual appeal, causal trees suffer from several well-known limitations:

- **Instability:** Small changes in data can lead to large changes in tree structure.
- **High variance:** Single-tree methods can be noisy, especially with small sample sizes.
- **No inference:** Confidence intervals and hypothesis testing for estimated effects are not naturally supported.

These limitations motivate the development of ensemble methods such as Causal Forests, which we now discuss.

14.4 From Causal Trees to Causal Forests

Causal Forests (Wager and Athey, 2018) extend the causal tree framework by aggregating many causal trees using a methodology inspired by Breiman's Random Forests. The idea is to stabilize the individual tree estimates and enable valid inference for the conditional average treatment effect $\tau(x)$.

14.4.1 Forest Construction and Estimation

Each tree in the causal forest is trained on a bootstrap sample of the data and considers a random subset of covariates at each split (feature bagging). For a new observation x, the prediction proceeds as follows:

1. Each tree b defines a partition of the space and assigns x to a leaf $L_b(x)$.
2. Within each leaf, the treatment effect $\hat{\tau}^{(b)}(x)$ is computed as the difference in average outcomes for treated and control units.
3. The final forest prediction is the average over B trees:

$$\hat{\tau}_{\text{forest}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{\tau}^{(b)}(x)$$

14.4.2 Honest Estimation

A key technical contribution of Athey and Imbens (2016) and Wager and Athey (2018) is the idea of **honest estimation**. In traditional decision trees, the same data are used to determine both the split structure and the predicted values. This leads to **adaptive overfitting**, where the model exaggerates differences that arise purely by chance.

Honest estimation solves this by partitioning the data into two disjoint subsamples:

- A *splitting sample*, used to determine where to split the covariate space.
- An *estimation sample*, used to compute the treatment effect within each leaf.

Formal setup: Let $\mathcal{D} = \mathcal{D}_{\text{split}} \cup \mathcal{D}_{\text{est}}$. For each leaf L, the honest treatment effect estimator is:

$$\hat{\tau}_L^{\text{honest}} = \frac{1}{n_{L,1}^{\text{est}}} \sum_{i \in L \cap \mathcal{D}_{\text{est}}, D_i = 1} Y_i - \frac{1}{n_{L,0}^{\text{est}}} \sum_{i \in L \cap \mathcal{D}_{\text{est}}, D_i = 0} Y_i$$

Why honesty matters:

- It prevents information leakage from the training phase into estimation, preserving statistical validity.
- It allows for unbiased treatment effect estimation, even in adaptive settings.
- It enables asymptotic inference: root-n consistency and asymptotic normality.

14.4.3 *Inference and Theory*

Wager and Athey (2018) show that under standard conditions, the CATE estimator $\hat{\tau}(x)$ produced by a causal forest satisfies:

- **Consistency:** $\hat{\tau}(x) \to \tau(x)$ as $n \to \infty$;
- **Asymptotic normality:** for each x, $\sqrt{n}(\hat{\tau}(x) - \tau(x)) \xrightarrow{d} \mathcal{N}(0, \sigma^2(x))$;
- **Variance estimation:** the variance $\sigma^2(x)$ can be estimated using influence functions.

14.4.4 *Software and Practice*

Causal Forests are implemented in the open-source `grf` package in R, which supports:

- Honest splitting and estimation;
- Estimation of standard errors;
- Confidence intervals and policy evaluation routines.

In applied work, causal forests provide an appealing balance between flexibility, interpretability, and inferential rigor, making them a popular choice for modern HTE estimation.

14.5 Generalized Random Forests and the k-Nearest Neighbor Perspective

The Generalized Random Forest (GRF) framework, introduced by Athey et al. (2018), provides a unifying perspective that connects tree-based ensemble methods with nonparametric local estimation techniques such as kernel regression and k-nearest neighbor (kNN) methods.

14.5.1 GRF as Adaptive Local Estimators

GRFs estimate functions $\theta(x)$ that solve local moment equations of the form:

$$\mathbb{E}[\psi(Z; \theta(x)) \mid X = x] = 0$$

where ψ is a score function (e.g., residual, treatment indicator, loss gradient), and Z is the observed data vector. For HTE, this reduces to estimating:

$$\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]$$

GRFs construct adaptive weights $\alpha_i(x)$ for each training example, which capture how "close" unit i is to x in terms of feature similarity, as learned through the forest structure. Then the estimated parameter is:

$$\hat{\tau}(x) = \sum_{i=1}^{n} \alpha_i(x) \cdot \hat{\tau}_i$$

This formulation closely resembles k-nearest neighbors, but with:

- **Adaptive neighborhoods**: defined by forest leaves rather than Euclidean distance.
- **Data-driven weighting**: units in similar leaves contribute more to the estimation.

14.5.2 Key Features of GRF

- GRF supports **continuous, binary, or multivalued treatments**.
- It generalizes beyond treatment effect estimation to quantile regression, instrumental variables, and policy evaluation.
- Provides automatic variance estimation via influence functions.
- Inherits valid inference from the honesty and subsampling structure of causal forests.

14.5.3 GRF vs Causal Forests

While causal forests focus specifically on $\tau(x)$ for binary treatments, GRF is a broader class of methods. Causal Forest is a special case of GRF where the score function ψ corresponds to residualized treatment effects.

Implementation: The R package `grf` implements GRF for a wide range of causal and predictive tasks.

14.6 Evaluating HTE Estimators

Once an HTE estimator has been built, it is crucial to assess its quality. Unlike ATE estimation, where performance can be judged through bias and variance, HTE estimation involves the evaluation of *functions*, making the task more complex.

14.6.1 Ground Truth CATE is Rarely Observed

In practice, we cannot observe both $Y(1)$ and $Y(0)$ for the same individual, so the "true" $\tau(x)$ is unobservable. This makes direct validation of CATE estimates infeasible without simulated data or experimental benchmarks.

14.6.2 Two Key Evaluation Criteria

1. Ranking Accuracy Suppose we use $\hat{\tau}(x)$ to rank individuals for treatment assignment. We want this ranking to reflect the true ordering of treatment benefits. This motivates evaluation via:

- **Cumulative Gain Curves**: plotting the total gain in outcomes when treating the top-k percentile units by predicted CATE.
- **Qini Coefficient / Uplift AUC**: analogs of ROC-AUC for treatment effects.

Example for Gain Curves Consider an RCT with 100 units divided into 5 quantiles based on predicted CATE:

- Predicted Quantiles: [90%-100%], [80%-90%], [70%-80%], [60%-70%], [50%-60%]
- Treatment Sample Sizes (N_i^T): [20, 20, 20, 20, 20]
- Control Sample Sizes (N_i^C): [20, 20, 20, 20, 20]
- Treatment Outcomes (Y_i^T): [0.9, 0.8, 0.7, 0.6, 0.5]
- Control Outcomes (Y_i^C): [0.3, 0.4, 0.5, 0.6, 0.7]

The formula from calculating the cumulative gain at the top-t quantiles is:

$$\text{CumulativeGain}(t) = \left(\frac{\sum_{i=1}^{t} Y_i^T}{\sum_{i=1}^{t} N_i^T} - \frac{\sum_{i=1}^{t} Y_i^C}{\sum_{i=1}^{t} N_i^C} \right) \times \left(\sum_{i=1}^{t} N_i^T + \sum_{i=1}^{t} N_i^C \right)$$

By plugging in the data from the RCT, we can calculate the cumulative gain top-t quantile as follows

- At $t = 1$: $\left(\frac{0.9}{20} - \frac{0.3}{20}\right) \times 40 = 0.6 \times 2 = 1.2$
- At $t = 2$: $\left(\frac{0.9+0.8}{40} - \frac{0.3+0.4}{40}\right) \times 80 = \left(\frac{1.7}{40} - \frac{0.7}{40}\right) \times 80 = 1 \times 2 = 2$

2. Calibration and Fit Some methods estimate the entire function $\hat{\tau}(x)$ over x. One can assess:

- **Calibration**: are predicted effects consistent with empirical outcomes in grouped data?
- **Fit metrics**: e.g., mean squared error of CATE in synthetic datasets where true $\tau(x)$ is known.

14.6.3 Simulation-based Evaluation

In simulation studies (e.g., DGPs from Hill, 2011; Künzel et al., 2019), one can directly compare $\hat{\tau}(x)$ to the true $\tau(x)$:

$$\text{MSE}(\hat{\tau}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{\tau}(x_i) - \tau(x_i))^2$$

14.6.4 Empirical Validation via Policy Evaluation

In empirical settings, a common approach is to evaluate the estimated CATE indirectly through its impact on **policy decisions**. For a policy function:

$$\pi(x) = \mathbb{I}[\hat{\tau}(x) > 0]$$

one can estimate the average effect of assigning treatment according to $\pi(x)$ and compare it with baseline or random assignment.

Note: This links HTE evaluation with treatment targeting problems and personalized decision-making.

14.6.5 Summary

Evaluating HTE estimators requires both:

- Functional assessment (calibration, smoothness, ranking),
- Behavioral assessment (how good are decisions based on $\hat{\tau}(x)$?).

Proper evaluation often involves a mixture of simulations, cross-validation, and domain-specific diagnostics.

14.7 Meta-Learners for HTE Estimation

Meta-learners are a modular class of algorithms designed to estimate conditional average treatment effects (CATEs) using existing supervised learning models. Instead of directly targeting causal structure (as in causal forests or DML), meta-learners decompose the estimation task into sub-problems solvable via any machine learning algorithm.

Introduced and formalized by Künzel et al. (2019), meta-learners are model-agnostic and easy to implement. They are particularly popular in practical applications and industry environments where interpretability, scalability, and ease of use are essential.

Problem Setup

Let $\{(X_i, D_i, Y_i)\}_{i=1}^n$ denote i.i.d. data where:

- $X_i \in \mathbb{R}^p$ are covariates,
- $D_i \in \{0, 1\}$ is a binary treatment indicator,
- Y_i is the observed outcome.

We aim to estimate:

$$\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]$$

T-Learner

The **T-learner** fits two separate models:

$$\mu_1(x) = \mathbb{E}[Y \mid X = x, D = 1], \quad \mu_0(x) = \mathbb{E}[Y \mid X = x, D = 0]$$

Then computes:

$$\hat{\tau}(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$$

Pros: Flexible, works well when treated and control groups are well-separated. *Cons:* Inefficient when data are imbalanced; fails to borrow information across groups.

S-Learner

The **S-learner** fits a single model using treatment D as a feature:

$$\mu(x, d) = \mathbb{E}[Y \mid X = x, D = d]$$

Then predicts:

$$\hat{\tau}(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0)$$

Pros: Simple, unified model. *Cons:* May underfit CATE if model is not sensitive to D.

X-Learner

The **X-learner** uses imputation and weighting:

1. Estimate $\hat{\mu}_1(x)$, $\hat{\mu}_0(x)$ as in T-learner.
2. Impute individual-level effects for each group:

$$\hat{D}_i^{(1)} = Y_i - \hat{\mu}_0(X_i) \quad \text{for treated,} \quad \hat{D}_i^{(0)} = \hat{\mu}_1(X_i) - Y_i \quad \text{for control}$$

3. Fit models for imputed treatment effects in each group.
4. Combine using weighting by propensity score $e(x)$:

$$\hat{\tau}(x) = g(x) \cdot \hat{\tau}_1(x) + (1 - g(x)) \cdot \hat{\tau}_0(x)$$

Pros: Very flexible; performs well under imbalance. *Cons:* Slightly more complex; requires propensity estimation.

Remarks

- Meta-learners do not require strong structural assumptions.
- They work best when paired with flexible base learners (e.g., random forests, neural nets).
- However, they lack built-in mechanisms for inference (e.g., confidence intervals).

14.8 HTE Estimation for Policy Targeting

A core application of CATE estimation is personalized decision-making: who should be treated? In the presence of heterogeneous treatment effects, optimal resource allocation requires more than knowing whether the average treatment effect is positive—it requires knowing where treatment is *most effective*.

14.8.1 Policy Function Based on $\hat{\tau}(x)$

Suppose we have a binary treatment and a limited budget to treat only a subset of the population. We can construct a **treatment rule**:

$$\pi(x) = \mathbb{I}[\hat{\tau}(x) > \theta]$$

where θ is a threshold determined by budget constraints, fairness criteria, or opportunity cost.

The resulting policy rule assigns treatment only to individuals expected to benefit the most, according to the estimated CATE.

14.8.2 Optimal Treatment Assignment

Given a budget constraint $\mathbb{E}[\pi(x)] \leq \rho$, the optimal policy is to treat the top ρ fraction with the highest estimated $\hat{\tau}(x)$. This can be implemented via:

- CATE ranking
- Thresholding with estimated quantiles
- Value-based or constrained optimization

14.8.3 Evaluating Policies

The value of a treatment policy π is the expected gain from following the policy:

$$V(\pi) = \mathbb{E}\left[Y(1) \cdot \pi(X) + Y(0) \cdot (1 - \pi(X))\right]$$

Under unconfoundedness, this can be estimated using doubly robust methods such as:

- Augmented IPW (AIPW)
- Inverse probability weighting
- Model-based plug-in with cross-fitting

14.8.4 Targeting and Fairness

HTE-based policies raise fairness concerns:

- Are disadvantaged groups disproportionately excluded?
- Does the treatment rule satisfy equal opportunity or demographic parity?

Solutions include fairness-aware CATE models or applying group constraints during targeting.

14.9 Practical Guidelines for Choosing HTE Estimators

There is no universally optimal method for estimating heterogeneous treatment effects. The appropriate choice depends critically on the researcher's specific goals, the nature and quality of available data, and the desired trade-offs among interpretability, statistical rigor, and implementation complexity. In this section, we offer practical guidance on selecting an HTE estimation strategy under different analytical objectives.

14.9.1 *Choosing Based on Analytical Goals*

When the primary goal is **interpretability**—for example, identifying subgroups for policy design or exploratory analysis—then *Causal Trees* and related *honest tree-based models* are often preferred. These methods produce human-readable partitions of the covariate space, making them ideal for stakeholder communication or theoretical hypothesis generation. However, their statistical properties are limited: they typically lack formal inference guarantees and may be unstable in small samples.

If the analyst is instead focused on **statistical precision and valid inference**, especially in high-dimensional settings, then methods such as *Causal Forests*, *Generalized Random Forests (GRF)*, or *Double Machine Learning (DML)* are more appropriate. These estimators are designed to deliver root-n consistent and asymptotically normal estimates of the CATE. They often incorporate sample splitting (honesty or cross-fitting) and regularization to avoid overfitting, while also allowing for construction of confidence intervals. Their complexity is higher, but they provide robust estimation under clearly defined assumptions.

Finally, when the main use case involves **personalized decision-making at scale**—for instance, targeted marketing, recommendation systems, or real-time decision engines—then *meta-learners* (such as the T-, S-, and X-learner families) and *uplift modeling* frameworks are commonly used. These methods are model-agnostic and easily combined with standard machine learning pipelines (e.g., neural networks, gradient boosting). While they excel in scalability and flexibility, they often do not provide built-in tools for inference, and their theoretical guarantees are weaker compared to GRF or DML.

14.9.2 *Implementation Advice and Caveats*

Across all methods, several practical considerations should be kept in mind. First, it is essential to verify the *overlap assumption*—that is, that each unit has a positive probability of receiving both treatment and control—and to assess covariate balance across treatment groups. Violations of this assumption undermine identification and may bias results, regardless of the estimator used.

Second, one should consider adopting *honest estimation strategies*, especially in tree-based methods. By separating the data used for model construction from the data used for estimation, honesty reduces overfitting and enables more reliable inference.

Third, where feasible, researchers should conduct *cross-validation*, *simulation-based model selection*, or *sensitivity analysis* to validate estimator performance. This is especially important in observational settings where ground truth is unknown.

Finally, if statistical inference—such as confidence intervals or hypothesis testing—is required, one should favor estimators with strong asymptotic properties, such as GRF or DML, over more flexible but heuristic-based methods like meta-learners.

In summary, the choice of HTE estimator should be driven by the research goal and context: causal trees for interpretability, forests and DML for inference, and meta-learners for large-scale personalization. No single method dominates in all settings, but thoughtful selection based on strengths and limitations can substantially improve both the credibility and utility of treatment effect estimates.

Table 13. Summary: Choosing Among HTE Estimators

Estimator	**Best For**	**Caveats**
Causal Tree	Interpretation, subgroup discovery	No inference; high variance and sensitivity to small data changes
Causal Forest / GRF	Inference in high-dimensional, flexible settings	Requires careful implementation; assumes unconfoundedness and honesty
Double Machine Learning	Theoretical rigor; bias reduction in high-dimensional models	Requires sample splitting; more complex setup; often focused on ATE
Meta-learners (T/S/X)	Scalability; plug-and-play with ML models	Lacks native inference; performance depends on base learner quality

14.10 The Notebooks

- This notebook DML_EconML_CATE estimates the conditional average treatment effects (CATEs);
- This notebook DML_EconML_Forest demonstrates the estimation of HTE (HTEs);
- This notebook EconML_vs_DoubleML compares the EconML Package, developed by Microsoft, with the DoubleML Package, developed by the original authors of (Chernozhukov et al., 2018).

Chapter 5: Appendices

15 Appendix A: Mathematical Prerequisites for Machine Learning

Machine learning, particularly supervised learning methods, builds heavily upon several foundational areas of mathematics. This appendix provides a more detailed review of the key concepts necessary for a full comprehension of subsequent material.

15.1 Linear Algebra

Linear algebra forms the backbone of most modern machine learning algorithms. It provides a concise way to represent and manipulate data, and it underpins many of the computational techniques used in ML.

15.1.1 Vectors and Matrices

- **Vectors:** A vector $x \in \mathbb{R}^d$ is a d-dimensional column of real numbers. It can be represented as:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

  where x_i are the elements of the vector. Vectors are fundamental for representing data points, features, and model parameters.

- **Matrices:** A matrix $X \in \mathbb{R}^{n \times d}$ is a rectangular array of numbers arranged in n rows and d columns. It can be represented as:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}$$

  where x_{ij} represents the element in the i-th row and j-th column. Matrices are used to represent datasets, linear transformations, and relationships between variables.

15.1.2 Operations

- **Matrix Multiplication:** If $X \in \mathbb{R}^{n \times d}$ and $\beta \in \mathbb{R}^d$, then the matrix product $X\beta \in \mathbb{R}^n$ is defined as:

  $$(X\beta)_i = \sum_{j=1}^{d} x_{ij}\beta_j$$

  where $(X\beta)_i$ is the i-th element of the resulting vector. Matrix multiplication is crucial for applying linear transformations to data and computing model predictions.

- **Inner Product (Dot Product):** For vectors $x, y \in \mathbb{R}^d$, the inner product (or dot product) is a scalar value defined as:

  $$\langle x, y \rangle = x^\top y = \sum_{i=1}^{d} x_i y_i$$

  The inner product measures the similarity or correlation between two vectors.

15.1.3 Norms

- **Euclidean Norm (L2 Norm):** The Euclidean norm of a vector x is its length, calculated as:

  $$\|x\|_2 = \sqrt{\sum_{i=1}^{d} x_i^2}$$

  The Euclidean norm is commonly used to measure the magnitude of a vector.

15.1.4 Important Matrix Properties

- **(AB) Transpose:** The transpose of the product of two matrices A and B is the product of the transposes of B and A in reverse order:

  $$(AB)^\top = B^\top A^\top$$

- **Symmetric Matrices:** A matrix X is symmetric if $X^\top = X$. The matrix $X^\top X$ is always symmetric and positive semi-definite, a property that is important in many optimization problems in machine learning. A matrix is positive semi-definite if for any non-zero vector z, $z^T X z \geq 0$.

15.1.5 Inverse and Pseudoinverse

- **Inverse:** A square matrix A is invertible (or non-singular) if there exists a matrix A^{-1} such that:

  $$AA^{-1} = A^{-1}A = I$$

  where I is the identity matrix. The inverse matrix ”undoes” the transformation of A.

- **Pseudoinverse:** If a matrix A is not invertible (e.g., it’s not square or it’s singular), we can use the Moore-Penrose pseudoinverse A^+. The pseudoinverse generalizes the concept of the inverse and provides a way to solve linear equations even when an exact inverse doesn’t exist. It is particularly important when dealing with overdetermined systems (more equations than unknowns) or underdetermined systems (more unknowns than equations).

15.1.6 Application in ML

- **Solving Linear Regression:** In linear regression, the optimal parameters $\hat{\beta}$ that minimize the sum of squared errors can be found using linear algebra:

  $$\hat{\beta} = (X^\top X)^{-1} X^\top Y$$

  where X is the matrix of input features, and Y is the vector of target variables.

- **Feature Transformations and Projections:** Linear algebra is used extensively for feature transformations (e.g., scaling, rotation) and dimensionality reduction techniques like Principal Component Analysis (PCA), where data is projected onto lower-dimensional subspaces.

15.2 Probability Theory and Statistics

Machine learning models often make probabilistic assumptions about the data generation process. Probability theory and statistics provide the tools to reason about uncertainty, model data distributions, and evaluate model performance.

15.2.1 Random Variables

- A random variable X is a variable whose possible values are numerical outcomes of a random phenomenon. It maps outcomes from a sample space to real numbers.

15.2.2 Expectation and Variance

- **Expectation (Expected Value):** The expected value (or mean) of a random variable X, denoted as $\mathbb{E}[X]$, is the average value of X over many trials. For a discrete random variable X, it is calculated as:

  $$\mathbb{E}[X] = \sum_x x\mathbb{P}(X = x)$$

  where the sum is taken over all possible values of X, and $\mathbb{P}(X = x)$ is the probability of X taking the value x.

- **Variance:** The variance of a random variable X, denoted as $\text{Var}(X)$, measures the spread or dispersion of its values around the mean. It is defined as:

  $$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

  A higher variance indicates that the values of X are more spread out.

15.2.3 Conditional Expectation

- **Conditional Expectation:** The conditional expectation of a random variable Y given another random variable X, denoted as $\mathbb{E}[Y|X]$, represents the average value of Y for a specific value of X. It is a function of X and is crucial in understanding how the expected value of one variable changes with the values of another.

15.2.4 Law of Large Numbers

- **Law of Large Numbers (LLN):** The Law of Large Numbers states that as the number of independent and identically distributed (i.i.d.) random samples n approaches infinity ($n \to \infty$), the sample mean converges to the true population mean. This principle justifies using sample averages to estimate population expectations.

15.2.5 Application in ML

- **Loss Minimization:** Many machine learning algorithms aim to minimize a loss function, which often corresponds to estimating conditional expectations. For example, minimizing the squared error loss in regression is equivalent to estimating the conditional expectation of the target variable given the input features.

- **Probabilistic Models:** Probabilistic models like logistic regression and Bayesian networks rely heavily on probability theory to model the relationships between variables and make predictions.

15.3 Optimization

Many machine learning models are trained by solving optimization problems. The goal is to find the set of parameters that minimizes a given cost or loss function.

15.3.1 Unconstrained Optimization

- **Unconstrained Optimization:** The general problem is to find the value of $\theta \in \mathbb{R}^d$ that minimizes a function $f(\theta)$, where there are no constraints on the values that θ can take.

15.3.2 First-Order Condition

- **First-Order Condition:** A necessary condition for a point θ^* to be a local minimizer of a differentiable function $f(\theta)$ is that the gradient of f at θ^* is zero:

  $$\nabla f(\theta^*) = 0$$

  This condition identifies stationary points, which may be minima, maxima, or saddle points.

15.3.3 Gradient Descent Algorithm

- **Gradient Descent:** Gradient descent is an iterative optimization algorithm that updates the parameter θ in the direction opposite to the gradient of the function $f(\theta)$ at the current point:

  $$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla f(\theta^{(k)})$$

  where $\theta^{(k)}$ is the value of θ at the k-th iteration, and $\eta > 0$ is the learning rate, which controls the step size.

15.3.4 Convex Functions

- **Convex Functions:** A function f is convex if for any two points x and y and any $\lambda \in [0, 1]$, the following inequality holds:

  $$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

Geometrically, this means that the line segment connecting any two points on the graph of f lies above or on the graph. Convex functions are important in optimization because any local minimum of a convex function is also a global minimum.

15.3.5 Application in ML

- **Minimizing Empirical Loss:** In machine learning, model training is often formulated as minimizing the empirical loss, which is the average loss over the training data. Gradient descent and its variants are widely used to solve these optimization problems.
- **Solving Logistic Regression:** Logistic regression, a common classification algorithm, is typically solved using convex optimization techniques to find the parameters that maximize the likelihood of the observed data.

15.4 Loss Functions

Loss functions quantify the discrepancy between a model's predictions and the true outcomes. They are essential for training machine learning models, as the goal is to find model parameters that minimize the loss.

15.4.1 Squared Loss (Regression)

- **Squared Loss:** For regression problems, where the goal is to predict a continuous output, the squared loss is commonly used:

  $$\ell(Y, \hat{Y}) = (Y - \hat{Y})^2$$

  where Y is the true outcome, and $\hat{Y}$ is the predicted outcome. The squared loss penalizes large errors more heavily than small errors.

15.4.2 Logistic Loss (Classification)

- **Logistic Loss (Cross-Entropy Loss):** For binary classification problems, where the goal is to predict the probability of an instance belonging to a certain class, the logistic loss (also known as cross-entropy loss) is used:

  $$\ell(Y, \hat{p}) = -Y \log(\hat{p}) - (1 - Y) \log(1 - \hat{p})$$

  where $Y \in \{0, 1\}$ is the true class label, and $\hat{p}$ is the predicted probability of $Y = 1$.

15.4.3 0-1 Loss (Classification, Theoretical)

- **0-1 Loss:** The 0-1 loss is a natural loss function for classification, which simply counts the number of misclassifications:

$$\ell(Y, \hat{Y}) = \mathbb{1}\{Y \neq \hat{Y}\}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function, which equals 1 if the condition inside is true, and 0 otherwise. While intuitive, the 0-1 loss is non-convex and discontinuous, making it difficult to optimize directly. It is primarily used for theoretical analysis and evaluation.

15.4.4 Application in ML

- **Model Training:** Model training is formulated as minimizing an aggregate loss function, which is the average (or sum) of the loss over the training data. The choice of loss function is crucial for the performance of a machine learning model.
- **Problem-Specific Losses:** Different machine learning problems require appropriately chosen loss functions. For example, squared loss is suitable for regression, logistic loss for binary classification, and other losses are designed for multi-class classification, ranking, and other specialized tasks.

16 Appendix B: Some Missing Proof

Proof of Theorem 12.2. Defining potential outcome residuals $\varepsilon_i(w) = Y_i(w) - \mathbb{E}_P[Y_i(w)]$ for $w = 0, 1$, we can express our estimation error as

$$\begin{aligned}\hat{\tau}_{DM} - \tau &= \frac{1}{n_1} \sum_{W_i=1} \varepsilon_i(1) - \frac{1}{n_0} \sum_{W_i=0} \varepsilon_i(0) \\ &= \frac{n}{n_1} \cdot \frac{1}{n} \sum_{i=1}^{n} W_i \varepsilon_i(1) - \frac{n}{n_0} \cdot \frac{1}{n} \sum_{i=1}^{n} (1 - W_i) \varepsilon_i(0).\end{aligned}$$

By randomization, one can verify that

$$\mathbb{E}[W_i\varepsilon_i(1)] = \mathbb{P}[W_i]\mathbb{E}[\varepsilon_i(1) \mid W_i = 1] = \mathbb{P}[W_i]\mathbb{E}[\varepsilon_i(1)] = 0, \quad \text{and} \quad \mathbb{E}[(1 - W_i)\varepsilon_i(0)] = 0,$$

and finally

$$\mathrm{Var}\left[\begin{pmatrix} W_i\varepsilon_i(1) \\ (1-W_i)\varepsilon_i(0) \end{pmatrix}\right] = \mathbb{E}\left[\left(\begin{pmatrix} W_i\varepsilon_i(1) \\ (1-W_i)\varepsilon_i(0) \end{pmatrix}\right)^{\otimes 2}\right]$$
$$= \begin{pmatrix} \pi \cdot \mathrm{Var}[\varepsilon_i(1)] & 0 \\ 0 & (1-\pi)\cdot \mathrm{Var}[\varepsilon_i(0)] \end{pmatrix}.$$

Thus, by the standard multivariate central limit theorem,

$$\sqrt{n}\begin{pmatrix} \frac{1}{n}\sum_{i=1}^n W_i\varepsilon_i(1) \\ \frac{1}{n}\sum_{i=1}^n (1-W_i)\varepsilon_i(0) \end{pmatrix} \Rightarrow \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \pi \cdot \mathrm{Var}[\varepsilon_i(1)] & 0 \\ 0 & (1-\pi)\cdot \mathrm{Var}[\varepsilon_i(0)] \end{pmatrix}\right).$$

Together with the fact that $\frac{n_1}{n} \to_p \pi$, the result above implies that $\sqrt{n}(\hat{\tau}_{DM} - \tau) \xrightarrow{d} \mathcal{N}(0, V_{DM})$, which follows directly from the Slutsky's theorem. □

Proof of Theorem 12.3. By the Law of Large Numbers applied separately to the treated and control groups:

$$\frac{1}{n_1}\sum_{i:W_i=1}(Y_i(1) - \mathbb{E}[Y_i(1)])^2 \xrightarrow{p} \mathrm{Var}[Y_i(1)]$$

$$\frac{1}{n_0}\sum_{i:W_i=0}(Y_i(0) - \mathbb{E}[Y_i(0)])^2 \xrightarrow{p} \mathrm{Var}[Y_i(0)]$$

Thus:

$$\hat{V}_{\mathrm{DM}} \xrightarrow{p} V_{\mathrm{DM}}.$$

□

Proof of Corollary 12.4.1. If CIA is violated, there exists an unmeasured confounder U_i such that $\{Y_i(0), Y_i(1)\} \not\perp W_i \mid X_i$. Thus:

$$\mathbb{E}[Y_i(1) \mid W_i = 1, X_i] \neq \mathbb{E}[Y_i(1) \mid X_i], \quad \mathbb{E}[Y_i(0) \mid W_i = 0, X_i] \neq \mathbb{E}[Y_i(0) \mid X_i].$$

Adjustment methods like stratification compute:

$$\mathbb{E}[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i]],$$

which does not equal $\mathbb{E}[Y_i(1) - Y_i(0)]$ due to confounding. Similarly, IPW or matching fails as they rely on CIA to balance groups. Residual confounding by U_i biases all such estimates. □

Proof of Corollary 12.4.1. If $\mathbb{P}(W_i = 1 \mid X_i = x) = 0$, then $\mathbb{E}[Y_i \mid W_i = 1, X_i = x]$ is undefined, as no treated units exist for $X_i = x$. Similarly, if $\mathbb{P}(W_i = 1 \mid X_i = x) = 1$, then $\mathbb{E}[Y_i \mid W_i = 0, X_i = x]$ is undefined. For adjustment:

$$\text{ATE} = \mathbb{E}[\mathbb{E}[Y_i \mid W_i = 1, X_i] - \mathbb{E}[Y_i \mid W_i = 0, X_i]],$$

these undefined terms prevent computation over the full support of X_i. For IPW, if $e(X_i) = 0$ or 1, weights become undefined. Matching and stratification fail due to lack of comparable units. Thus, the ATE cannot be consistently estimated. □

References

Argyle, L. P. et al. (2024). Caution in using large language models as human surrogates: Experimental evidence from the evaluation of algorithmic decision-making. *arXiv preprint arXiv:2410.19599*.

Ash, E. and Hansen, S. (2023). Text algorithms in economics. *Annual Review of Economics*, 15(1):659–688.

Athey, S. and Imbens, G. W. (2016). Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360.

Athey, S., Tibshirani, J., and Wager, S. (2018). Generalized random forests.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

Belloni, A., Chernozhukov, V., and Hansen, C. (2014). Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies*, 81(2):608–650.

Belloni, A., Chernozhukov, V., and Wei, Y. (2016). Post-selection inference for generalized linear models with many controls. *Journal of Business & Economic Statistics*, 34(4):606–619.

Bi, M. and Team, D. (2024). Deepseek-v3 and r1: Open-source rl trained reasoning llms. *arXiv preprint arXiv:2401.06066*.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020a). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020b). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Card, D., Chang, S., Becker, C., Mendelsohn, J., Voigt, R., Boustan, L., Abramitzky, R., and Jurafsky, D. (2022). Computational analysis of 140 years of us political speeches reveals more positive but increasingly polarized framing of immigration. *Proceedings of the National Academy of Sciences*, 119(31):e2120510119.

Chang, N.-c. (2020). Double/debiased machine learning for difference-in-differences models. *Econometrics Journal*, 23(2):177–191.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W. K., and Robins, J. M. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68.

Chernozhukov, V., Hansen, C., Kallus, N., Spindler, M., and Syrgkanis, V. (2024a). Causal diagrams and identification. In *Applied Causal Inference: Powered by ML and AI*, chapter 12. CausalML-book.org. Available at https://causalml-book.org; arXiv:2403.02467.

Chernozhukov, V., Hansen, C., Kallus, N., Spindler, M., and Syrgkanis, V. (2024b). Causal diagrams and identification. In *Applied Causal Inference: Powered by ML and AI*, chapter 9. CausalML-book.org. Available at https://causalml-book.org; arXiv:2403.02467.

Chernozhukov, V., Risse, K., and Spindler, M. (2022). Automatic debiased machine learning of causal and structural effects. *Econometrica*, 90(3):1567–1606.

Corrigan, G. and Dube, J.-P. (2024). An applied econometric framework for large language models. *arXiv preprint arXiv:2412.07031*.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems (NeurIPS)*.

DeepSeek (2024). Deepseek-v3 technical overview. Available at https://deepseek.com/research/v3.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. (2025). Deepseek-v3 technical report.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient fine-tuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dubé, A., Jacobs, J., Naidu, S., and Suri, S. (2023). Monopsony in online labor markets. *AEA Papers and Proceedings*, 113:292–296.

Farrell, M. H., Liang, T., and Misra, S. (2020). Deep learning for individual heterogeneity: An automatic inference framework. *arXiv preprint arXiv:2010.14694*. University of Chicago, Booth School of Business.

Farrell, M. H., Liang, T., and Misra, S. (2021). Deep neural networks for estimation and inference. *Econometrica*, 89(1):181–213.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Sciences*, volume 55, pages 119–139. Elsevier.

Fuhr, J., Berens, P., and Papies, D. (2024). Estimating causal effects with double machine learning—a method evaluation. *arXiv preprint arXiv:2403.14385*.

Gentzkow, M., Kelly, B., and Taddy, M. (2019). Text as data. *Journal of Economic Literature*, 57(3):535–574.

Gordon, B. R., Moakler, R., and Zettelmeyer, F. (2023). Close enough? a large-scale exploration of non-experimental approaches to advertising measurement. *Marketing Science*, 42(6):1064–1083.

Gorodnichenko, Y., Pham, T., and Talavera, O. (2023). The voice of monetary policy. *American Economic Review*, 113(2):548–84.

Hansen, S., Lambert, P. J., Bloom, N., Davis, S. J., Sadun, R., and Taska, B. (2023). Remote work across jobs, companies, and space. Technical report, National Bureau of Economic Research.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 1026–1034.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, D., et al. (2021). Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Hill, J. L. (2011). Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, L., and Chen, W. (2022). Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing*. Pearson Prentice Hall.

Künzel, S. R., Sekhon, J. S., Bickel, P. J., and Yu, B. (2019). Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10):4156–4165.

Li, Z., Wang, F., Yuan, H., Tian, X., Liu, Y., Qian, C., Sun, M., Shi, Y., and Xie, J. (2024). Native sparse attention: Scaling efficient transformers with hardware-aware sparsity. *arXiv preprint arXiv:2502.11089*.

Lin, X. et al. (2024). Oasas: Social interaction simulations with one million agents. *arXiv preprint arXiv:2411.11581*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Liu, Z., Huang, D., Huang, K., Li, Z., and Zhao, J. (2021). Finbert: A pre-trained financial language representation model for financial text mining. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 4513–4519.

LMSYS (2023). Chatbot arena: An open platform for evaluating llms with human preferences. https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. *International Conference on Learning Representations (ICLR)*.

Manzoor, E., Chen, G. H., Lee, D., and Smith, M. D. (2023). Influence via ethos: On the persuasive power of reputation in deliberation online. *Management Science*, 70(3):1613–1634.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

McKenzie, A., Zhang, K., Lee, A., and Liu, B. (2025). Simple test-time scaling for reasoning in large language models. *arXiv preprint arXiv:2504.01234*.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. (2018). Mixed precision training. In *International Conference on Learning Representations (ICLR)*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Neyman, J. (1959). Optimal asymptotic tests of composite statistical hypotheses. *Probability and Statistics: The Harald Cramér Volume*, pages 213–234.

OpenAI (2023). Learning from human feedback. *OpenAI Technical Report*. https://openai.com/research/learning-from-human-feedback.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, D., Liang, P., Bernstein, M. S., and Heer, J. (2023). Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Quinlan, J. R. (1986). Induction of decision trees. In *Machine Learning*, volume 1, pages 81–106. Springer.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018a). Improving language understanding by generative pre-training. OpenAI Blog.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018b). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog.

Rafailov, R., Liu, X., Zhang, Y., Yang, Y., and Hashimoto, T. (2023). Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Reisenbichler, M., Reutterer, T., Schweidel, D. A., and Dan, D. (2022). Frontiers: Supporting content marketing with natural language generation. *Marketing Science*, 41(3):441–452.

Robinson, P. M. (1988). Root-n-consistent semiparametric regression. *Econometrica: Journal of the Econometric Society*, pages 931–954.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanik, J., and Szczepinska, M. (2021). Detecting formal thought disorder by neural embedding of speech. *NPJ Schizophrenia*, 7(1):1–9.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*.

Shi, R., Tian, X., Yang, M., and Li, Z. (2024). What, why, and how: An empiricist's guide to double/debiased machine learning. *SSRN Working Paper 4677553*.

Shinn, N., Labash, E., Lee, Y., Chowdhery, A., Shachaf, I., and Zeng, A. (2023). Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Soldaini, L., Kinney, R., Bhagia, A., Schwenk, D., Atkinson, D., Authur, R., Bogin, B., Chandu, K., Dumas, J., Elazar, Y., Hofmann, V., Jha, A. H., Kumar, S., Lucy, L., Lyu, X., Lambert, N., Magnusson, I., Morrison, J., Muennighoff, N., Naik, A., Nam, C., Peters, M. E., Ravichander, A., Richardson, K., Shen, Z., Strubell, E., Subramani, N., Tafjord, O., Walsh, P., Zettlemoyer, L., Smith, N. A., Hajishirzi, H., Beltagy, I., Groeneveld, D., Dodge, J., and Lo, K. (2024). Dolma: an open corpus of three trillion tokens for language model pretraining research.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Sutton, R. S. (2019). The bitter lesson. urlhttp://www.incompleteideas.net/IncIdeas/BitterLesson.html. Accessed: 2025-04-28.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wager, S. and Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523):1228–1242.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., et al. (2022). Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Weng, L. (2024). Reward hacking and misalignment in rlhf. https://lilianweng.github.io/posts/2024-01-01-reward-hacking/. Accessed: 2024-05-26.

Wiener, N. (1948). *Cybernetics: Or Control and Communication in the Animal and the Machine*. MIT Press.

Wikipedia (2023). Mixture of experts. https://en.wikipedia.org/wiki/Mixture_of_experts. Accessed: 2025-05-21.

Wikipedia contributors (2023). Graphics processing unit. https://en.wikipedia.org/wiki/Graphics_processing_unit. Accessed: 2025-05-22.

Xu, Y., Ghose, A., and Xiao, B. (2023). Mobile payment adoption: An empirical investigation of alipay. *Information Systems Research*. Published Online: 7 Jul 2023.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Yao, S., Zhao, D., Yu, J., Zhao, I., Weld, D. S., et al. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Yao, S., Zhao, J., Yu, D., Narasimhan, K., Jiang, Y., Cao, Q., Kasai, J., Wang, Y., Bosselut, A., and Zettlemoyer, L. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Ye, Z., Zhang, Z., Zhang, D. J., Zhang, H., and Zhang, R. (2023). Deep-learning-based causal inference for large-scale combinatorial experiments: Theory and empirical evidence. arXiv preprint arXiv:2306.17559. Available at SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4485708.

Zhang, P., Wu, A., Deng, L., and Chi, E. (2024). Ai-augmented estimation: Low-bias, low-variance inference with llm assistance. *arXiv preprint arXiv:2404.11111*.